AN ARCHITECTURE FOR DYNAMIC META-LEVEL PROCESS CONTROL FOR MODEL-BASED TROUBLESHOOTING

THESIS

John E. Friskie, Captain, USAF

AFIT/GCE/ENG/95D-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/95D-02

AN ARCHITECTURE FOR DYNAMIC META-LEVEL PROCESS CONTROL FOR MODEL-BASED TROUBLESHOOTING

THESIS

John E. Friskie, Captain, USAF

AFIT/GCE/ENG/95D-02

AFIT/GCE/ENG/95D-02

AN ARCHITECTURE FOR DYNAMIC META-LEVEL PROCESS CONTROL FOR MODEL-BASED TROUBLESHOOTING

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

John E. Friskie, B.S., E.E.

Captain, USAF

December 1995

ACKNOWLEDGMENTS

I would like to express my thanks to those who helped me complete my thesis work. First, thanks go to my family: my wife Vicki, and my kids Vanessa, Jimmy and newly arrived Alyxandria. Vicki: Thanks for pushing me to get here in the first place and sustaining me the whole time. Vanessa and Jimmy: Thanks for allowing me to be "Ghost Dad" for eighteen months, and for taking up your share of the things I normally do at home. Alyxandria: You haven't done much but coo and goo yet, but your smile to me when I came home from work made the rough day worth it. Everyone: A special thanks for your help during the last few weeks of crunch time.

Second, I would like to thank my thesis advisor Maj (Lt Col select) Gregg Gunsch. Our weekly meetings ranged from ten minute status reports to two hour tutorials on how to use the Internet. When I thought my research path was going nowhere, he would assure me I was still doing all right. When I had no idea what an article was trying to get across (and the article was invariably related to my research), his insights got me going. His real world examples of fixing TV's, microwaves, and car alarms gave me a new perspective in which to see how my research could go beyond academia and often gave me the practical thoughts I needed to make sense of what we discussed.

Third, thanks to my other committee members Dr. Eugene Santos and Maj Keith Shomper. Although he doesn't know it, Dr. Santos' casual mentioning of "meta level" opened the door to my thesis architecture. Maj Shomper's guidance as academic advisor and patience with answering my many questions from different classes is greatly appreciated.

Last, thanks to my classmate, good friend, and cookout partner Kevin Anchor. He helped me get rid of the "deer caught in headlights" syndrome I suffered in more than a few classes we shared. I hope I can return the favor someday.

John E. Friskie

Table of Contents

List of Figures

List of Tables

Abstract

There are numerous methods used for troubleshooting devices. Each method has certain domains, knowledge requirements, and assumptions required for it to perform well. However, oftentimes no one method by itself is sufficient to completely solve a troubleshooting problem. Therefore, an architecture is required to control the combined use of many problem solving methods. The combination of multiple problem solving methods makes the troubleshooting process more robust in terms of device domains that can be dealt with and quality of diagnoses produced.

Troubleshooting has two tasks: diagnosis and problem resolution. This research provides an architecture that allows dynamic method selection during diagnosis. Dynamic method selection factors the current state of the diagnosis process along with other method parameters to determine which method to use to advance the diagnosis process.

The architecture was developed by combining themes from diagnosis research that focused on dynamic multimethod diagnosis and its control.

This work has produced several results. It provides an architecture to organize the methods and a basis for making control decisions concerning method use during diagnosis. It identifies a generous number of methods useful to perform diagnosis. It identifies the knowledge these methods require.

AN ARCHITECTURE FOR DYNAMIC META-LEVEL PROCESS CONTROL FOR MODEL-BASED TROUBLESHOOTING

I. Introduction

1.1 Chapter Overview

This chapter provides a high level description of the contents of this research. In doing so, it first describes the problem this research addressed. Next, it lists the objectives this research aimed to achieve. The specific scope the research focused on is discussed next, then follows the approach the research took in order to reach the objectives. Finally, the executive summary concisely restates the purpose, approach, and results for this thesis and outlines the remainder of the document.

1.2 Research Problem

1.2.1 Background. When an entity does not operate as its user expects, and the user desires the entity to be operational again, the *troubleshooting process* begins. Simply put, the overall goal of the troubleshooting process is to return an entity to operational status (16:11). The entity may be something physical like an electronic device or something abstract like a process. In realizing the overall goal of troubleshooting, a sub-goal must be realized: find a explanations for why the entity is not producing the expected results. The explanations can involve various things. For example, with a physical entity (like a device) perhaps some components of the device are broken; perhaps the settings that configure the device to operate for a certain function do not match the function the operator wished the device to perform; perhaps external forces the user is unaware of are influencing the device so that the operating assumptions are not being met. With an

abstract entity, like a process in an office, perhaps the resources needed to accomplish a task are not available as they were assumed to be. Whatever the entity, the explanations provide possible reasons for why it is misbehaving. The reasons contained in the explanations, in turn, help realize another sub-goal of the troubleshooting process: implement corrective actions that return the entity to correct operation. Here, the corrective actions aim to fix the reasons for misbehavior, such as replacing bad components, putting control dials to their proper settings, etc.

When working on the first sub-goal (finding explanations for misbehavior), a troubleshooter has basic jobs to accomplish. These are tasks. Furthermore, some tasks may be accomplished in more than one way. These are problem solving methods (1:44). A problem solving method dictates how a task can be accomplished. For each problem solving method, there are particular requirements to be met in order for the method to perform well or even perform at all. These requirements may state that particular knowledge must be available to the method or that particular assumptions hold regarding the problem the method is applied to. Also, these methods can be designed toward use in specific domains outside of which the method's results are invalid. Because of these requirements, oftentimes one problem solving method alone cannot find explanations for misbehavior in a troubleshooting problem. In these cases, to satisfy the first sub-goal of the troubleshooting process, it would be helpful for the different methods to work together to find explanations that they could not find by themselves.

1.2.2 Problem Statement. Therefore, the problem this research addresses is: how to make a generalized troubleshooting system that can match a given state of the troubleshooting process to method requirements and capabilities in order to choose the most appropriate method to advance the diagnostic reasoning.

1.3 Research Objectives

The purpose of this research was to investigate the following hypothesis: a *generalized troubleshooting system* can be designed which uses general troubleshooting heuristics and domain knowledge if available to guide the troubleshooting process. More specifically, this research set out to determine a generalized troubleshooting system architecture and theory of operation that could dynamically choose between competing problem solving methods when performing the first sub-goal of the troubleshooting process, namely finding explanations for a physical entity's misbehavior.

To achieve this objective, four sub-objectives were established:

1. Determine a robust set of problem solving methods in terms of domain and device independence;
2. Determine a robust set of knowledge the problem solving methods require;
3. Determine a basis for which heuristics can be developed to choose between problem solving methods;
4. Determine an organization into which the problem solving methods can be placed.

1.4 Scope

This thesis focused on a portion of the broad capabilities envision for a generalized troubleshooting system. Chapter three, section two discusses this vision. To obtain this portion, this thesis has been focused along two dimensions. The first dimension involves the troubleshooting process itself and the second involves the part of a larger project this thesis can contribute to.

Concerning the first focusing dimension, in order to have a workable piece of the problem this research addressed, three focusing assumptions were made. First, although

troubleshooting involves finding explanations and enacting corrective actions, this research focuses on the "finding explanations" sub-goal, hereafter called diagnosis. Second, the entities dealt with are physical rather that abstract. Third, this research focuses on using problem solving methods that use *model-based reasoning*. Assumptions two and three focus the type of systems usable with the generalized troubleshooting architecture to physical systems that can be characterized by interconnections of subsystems, which is true for nearly all engineered devices. These three assumptions allow for a generalized troubleshooting architecture to be developed without overly constraining the solution.

Just as the generalized troubleshooter examines a portion of the troubleshooting process, it also aims to be applicable to a portion of a larger program where troubleshooting is used. This program is MAGIC, or Multimission Advanced Ground Intelligent Control, a program to improve satellite control by using an intelligent real-time system. The program is managed by the Satellite Control and Simulation Division of Phillips Laboratory at Kirtland Air Force Base. The MAGIC vision is to "architect a ground station environment that will manage multiple missions, easily adapt to new missions, and improve operator effectiveness and enhance operational capabilities" (5:3). Part of this vision is to have an intelligent system capable of determining causes of satellite malfunctions and assisting the operator in devising solution plans. The generalized troubleshooter is applicable to this vision because of the range of device domains it can work on (multiple satellite missions and constellations) and its ability to propose possible causes for malfunction as the first step to proposing solution plans.

1.5 Approach

The approach used for this effort is divided into four phases. Phase one involved research into diagnosis in general and various model-based methods to do diagnosis. The model-based diagnosis systems were analyzed to find features among them that were

domain independent and to describe what knowledge and assumptions each system relied on. Phase two conducted research into other work involving a combined method approach to diagnosis. Phase three's task was to analyze the work of four separate research groups from phase two in order to find strengths in their architectures that were complementary. In phase four, an architecture and theory of operation were formed by combining the pieces gathered in phase three. Just as the combined method approach uses the strengths of different problem solving methods, phase four sought to develop a system that used the strengths of different combined method approaches.

1.6 Executive Summary

The executive summary concisely states the purpose of this research, its accomplishments, and provides a roadmap for the rest of the thesis.

1.6.1 Research Summary. This research has developed an architecture and theory of operation for an automated generalized troubleshooter. A generalized troubleshooter aims to be robust in terms of domains it can work on and knowledge it can use. Furthermore, a practical application of a generalized troubleshooter may lie in the MAGIC program's future goal of autonomous satellite control.

Many current automated diagnosis systems attempt to define *the* best approach to perform diagnosis. However, due to the variability of diagnosis problem characteristics, defining *the* best approach is too restrictive (25:1). Therefore, this thesis attempts to overcome the limitation of current single-approach diagnosis systems. To achieve robust performance, this work focused on the idea of many methods, each with their specialties, working together to solve a diagnosis problem. Additionally, the reasoning methods used in the generalized troubleshooter architecture use the model-based approach.

This work has produced several results. It identified a generous number of methods useful to perform the tasks involved in diagnosis. It identified the knowledge

these methods require. It provides an architecture to organize the methods and a basis for making control decisions concerning the method to use during the diagnosis process. Finally, it describes possible contributions an automated generalized troubleshooter could make to a significant Air Force goal: improved satellite control through automated anomaly detection and resolution.

1.6.2 Document Summary Chapter one provided a brief summary of the problem, objectives, scope and approach involved with this research. Chapter two provides details research results on the components used in the thesis architecture, namely the model-based problem solving methods, and work using a combined method approach to diagnosis. Chapter three discusses the approach to this thesis work, providing rationale for the scoping decisions, and presenting the organization of research work used to manage the project. Chapter four discusses in detail the generalized troubleshooter's design architecture and theory of operation. Chapter five analyzes the design architecture and theory of operation and outlines its strengths and areas for improvement. Finally, chapter 6 summarizes the research effort and provides recommendations for possible extensions to it.

II. Literature Review

2.1 Chapter Overview

This literature review presents past efforts in the field of automated diagnosis. After a brief discussion of the diagnosis part of troubleshooting, the first generation of automated diagnosis methods is presented. An analysis of their shortcomings motivates why the model-based approach is desirable in a generalized troubleshooter. Consequently, the elements from phase one of the research approach are then presented, namely concepts and methods from model-based diagnosis. Finally, results from phase two of the research approach are presented describing work involving a combined method approach to performing diagnosis

2.2 Diagnosis

In the next two subsections, diagnosis is discussed in regards to the main tasks it accomplishes and the elements that are used in accomplishing the tasks.

2.2.1 Diagnosis: the Process. According to The American Heritage Dictionary, to diagnose is "To perform an *examination* of (a person or thing)"; "To distinguish or *identify*..., as a disease" (17:326). Diagnosis is that part of the troubleshooting process where observations of device behavior are collected and used in *examining* device behavior in order to *identify* reasons for why a device is not behaving as expected.

In performing diagnosis, three main tasks are accomplished (1:43):

1. Symptom detection
2. Hypothesis Generation
3. Hypothesis Discrimination

Symptom detection involves collecting observations of a device's behavior and determines if an observation is indeed a symptom of device malfunction. It combines observations about the device, provided by some type of sensor, and knowledge of device operation to assign a "normal" or "abnormal" value to the observation. The output from the symptom detection task is an observation with a "normal" or "abnormal" value assigned to it. Those observations associated with an abnormal value are the observations that need further processing to identify reasons why the value is abnormal.

An interesting point is that symptom detection can produce different results depending on the point of view from which it is looked. For example, Mager states that a competent troubleshooter always verifies the symptoms reported before automatically starting to formulate hypotheses (16:39). This is often geared toward determining an operator error versus a device malfunction. For example, a printer would work fine if the baud rate switch was in the correct position or the monitor would produce a display if the brightness setting was not zero (16:34). In these cases, symptom detection from the operator's point of view would assign a value of abnormal to the device operation, whereas from the troubleshooter's point of view, considering the knowledge of device operation would lead him to not describe the device's operation as abnormal. Under these operator induced errors, the device itself works fine, but the operator's assumptions surrounding the operating environment of the device are not consistent with the actual operating environment. In these cases, a lengthy diagnosis process can be avoided when this type of problem is detected early. This difference in points of view is important because for this thesis the troubleshooter's point of view is taken which does not automatically consider an observation as abnormal and needing explanation, so by default symptoms are verified. This assumes the troubleshooter has more knowledge available to it to distinguish normal from abnormal operation.

Hypothesis generation involves determining an initial set of reasons for why the device has abnormal observations. For example, after performing some reasoning about the observations and the device operation, a proposed reason (hypothesis) can contain a set of components believed to be operating abnormally. If the components in the hypothesis are operating abnormally as the hypothesis suggests, then their misbehavior can account for the abnormal observations.

Hypothesis discrimination takes the hypotheses generated by the hypothesis generation step and decides which hypotheses to keep, refine, or reject, thereby pruning the list of hypotheses under consideration. These decisions can be based on additional knowledge such as additional behavior observations, knowledge of the likeliness of a hypothesis, or the cost of implementing a repair plan that would clear a fault identified in a hypothesis (if that knowledge is considered) to name just a few (12:2). This pruning narrows the scope of what is considered as causing the abnormal observations and eliminates some hypotheses from needing further consideration.

2.2.2 Elements of Diagnosis. The previous section described at a high level the "what" that is done during the diagnosis process. To accomplish these tasks, the diagnosis process needs to act on elementary objects (22:131) to transform the diagnosis state from supposing something is wrong with a device to proposing what may be wrong with the device.

The elements have already been presented, although not explicitly. The first element is *observations* of device behavior. Observations are the inputs to the diagnosis process. The second is *hypotheses* for why a device is not behaving correctly. Hypotheses are the outputs of the diagnosis process. In between, in transforming observations to hypotheses, *inferences* (reasoning from section 2.2.1) and *knowledge* are used. Inferences draw conclusions about possible reasons for faulty behavior based on the observations and knowledge. A major point of this thesis is that the way to perform these

inferences varies and can be controlled to solve a diagnosis problem. Finally, knowledge is needed for feeding the way inferences are performed. A brief list of the types of knowledge that could be used in the diagnosis process contains knowledge of: structure, function, correct behavior, faulty behavior, diagnostic hypotheses, component failure probabilities, causal models, available problem solving methods, selection criteria for these methods, historical data, assumptions of purpose the diagnosis session seeks to fulfill, and many more. A large part of this work was to find the types of knowledge that could be applied during diagnosis and how to fit the knowledge and the inference methods they relate to into the overall architecture for dynamic diagnosis.

2.2.3 Diagnosis Summary. In summary, diagnosis is the process of using knowledge and observations to infer hypotheses. It is a "...complex, non-monotonic process..." (23:412) where the three tasks of diagnosis may repeat during the same diagnosis session.

2.3 "First Generation" Automated Diagnosis

Having discussed what diagnosis involves, this section present past techniques for performing automated diagnosis. These techniques are presented to analyze their contributions to the overall goal of developing a generalized troubleshooter and to provide a basis of comparison for the model-based methods chosen as the methods used for this thesis.

2.3.1 Diagnostics. According to Davis, diagnostics are test programs used "...to ensure that the device is capable of doing everything it's supposed to do" (7:5). As such, they verify if the intended functionality is present in a device but cannot propose reasons for malfunction if some functionality is missing. They may be useful as a front end

to the Symptom Detection module, providing observations of the device's behavior, but in and of themselves their purpose is not to propose hypotheses.

2.3.2 Rule-based Expert System. An automated diagnosis system based on rules is developed by collecting the empirical associations of faults to symptoms (and the reasoning steps a troubleshooter takes along the way) generated by expert troubleshooters familiar with a specific device. Because the associations are based on experience, a feature of this technique is the long time usually required to acquire the associations (7:6). Another characteristic of a rule-based diagnosis system is that for a new device, new experience needs to be gained with it thereby creating a new rule base for the device. This leads to a high cost for the rule-based system since much effort goes into devising a diagnosis system applicable to just one or very few devices (22:316). Additionally, if experience has not revealed a particular fault/symptom association a device can display, the rule base cannot report on that fault when the symptom eventually occurs.

2.3.3 Decision Trees. "Decision trees provide a simple and efficient way to write down the sequence of tests and conclusions needed to guide a diagnosis" (7:6). They are a way to record the results of the decision making process a troubleshooter made in reaching hypotheses by guiding which branches to take in the tree based on the answer to questions at decision nodes. An important feature of decision trees is that results of decisions are recorded and not the knowledge used in arriving at those decisions in the first place.

2.3.4 Analysis of "First Generation" Automated Diagnosis. This section looks at the features of the techniques used in first generation automated diagnosis systems to see how they can contribute to the development of a generalized troubleshooter in terms of the thesis objectives (Chapter one, section three). Specifically, their shortcomings will be discussed and the gaps left will provide motivation for using another technique to perform diagnosis, namely model-based reasoning.

Since diagnostics do not perform diagnosis but verify functionality, their role in a generalized troubleshooter is limited. As mentioned above, they may serve as a front end to the diagnosis process, but do not present a means of generating hypotheses.

Rule-based diagnosis systems are efficient for a narrow domain; however, a generalized troubleshooter aims to be applicable to an array of domains. Because of this, one rule base is not usually general enough for many device domains. Additionally, rule-based diagnosis systems may not be able to provide a hypothesis because experience has not involved a new fault/symptom association. A generalized troubleshooter should be able to use knowledge not based on experience to lead to a hypothesis even if the symptom was not encountered before.

Last, decision trees record results of reasoning and not what was considered in doing the reasoning in the first place. A generalized troubleshooter should be able to use knowledge a human troubleshooter uses about what problem solving method is good to use and why (among other knowledge) to *recreate* the reasoning process and not just follow recorded decisions.

Because of these shortcomings, another method of reasoning to perform diagnosis is needed. The method chosen for this thesis is based on model-based reasoning.

2.4 Model-Based Diagnosis

Chapter one listed several focusing assumptions made as part of this research. One assumption involved focusing the kinds of methods used to do diagnosis. The kinds of methods used here are methods based on model-based reasoning. Model-based reasoning's advantages over first generation diagnosis systems, combined with the abundant research conducted in the model-based diagnosis field, make the model-based

methods contribute well to the overall objectives of this research . This section discusses the fundamentals of model-based diagnosis concepts and methods.

2.4.1. Operation of Model-Based Diagnosis. This section presents at a fundamental level how a model-based system performs diagnosis.

2.4.1.1 Model-Based Diagnosis Basic Paradigm. First generation diagnosis methods relied on something other than the computer initially reasoning about what was wrong. A model-based diagnosis system aims to reach diagnoses by recreating on the computer the reasoning process a human would go through, by directly using the knowledge a human uses rather than just using the results of human reasoning fed to it. The knowledge available to a human troubleshooter is widely varied and this knowledge in part is what is represented in models. To do its job, the basic paradigm of model-based diagnosis is to compare observations of device behavior to the behavior predicted by the device models and, when prediction does not match observation, search the model for where the agreement breaks down.

Another term for model-based diagnosis is *deep reasoning* or *reasoning from first principles* as opposed to the term *shallow reasoning* associated with first generation diagnosis systems. The term shallow reasoning attempts to convey the connotation that little of the internal workings of a device are known in doing diagnosis. Instead, the empirical fault/symptom associations are the major source of knowledge. Deep reasoning, on the other hand, attempts to convey the connotation that knowledge about what is actually happening inside a device is known and used during diagnosis. It is this type of low level device knowledge that forms a part of what is modeled.

2.4.1.2 Elementary Model Content. Besides being characterized by the depth of knowledge about a device's internal operation, the knowledge in device models can also be characterized by its different types. Early research in model-based diagnosis

identified device models containing a spectrum of knowledge that has two types at its ends:

1. Knowledge of the individual behavior and structural interconnectivity of device components;
2. Knowledge of device behavior based on knowledge of the how the behavior of a component causes behavior in other components.

Furthermore, the structure and individual behavior end of the spectrum uses only knowledge of the correct behavior of the components - how they are supposed to work. The other end of the spectrum - the causal end - deals with only knowledge of how a component could fail and what behavior a failure in one component would induce in other components or observable parameters.

2.4.1.3 Fundamental Approaches to Model-Based Diagnosis. A model-based diagnosis system based on the first spectrum end operates as follows: Expected device operation is predicted based on the structure and behavior models. The behavior models determine component output based on its input, and the structure model carries the values from device input to components, from component to component, and from components to device outputs. Observations are collected of the device outputs or other measurable parameters and compared to the predictions. Any observations that do not agree with predictions based on the models of correct component behavior indicate the device is malfunctioning somewhere, since if the components were operating as their correct behavior models say, then no contradictory observations would be produced. This approach to model-based diagnosis is often referred to as *consistency based diagnosis* because the normality or abnormality of observations is based on the consistency or inconsistency of those observations to their associated model predictions.

A model-based diagnosis system based on the second spectrum end operates differently. Here, observations deemed abnormal are fed to the diagnosis system and a

causal network is searched to find what malfunctions may have led to those observations. Since sometimes a single malfunction may have multiple results, only a certain set of malfunctions are proposed as diagnoses. That set is composed of the malfunctions that cover the abnormal observations and do not imply the device ought to be showing other observations that in fact the device is not showing. If a malfunction of a component predicts two observations need to be present and only one is observed, then this malfunction is pruned under this method because the results the model predicts for the proposed malfunction are not fully observed . This approach to model-based diagnosis is referred to as *abductive diagnosis* because of the abductive nature of the inferencing. Abductive inferencing is used to generate possible explanations and follows this process (20:1):

Rule: a CAUSES b

Given: b

Infer: a

For abductive diagnosis systems, b could be the abnormal observations and a could be a component malfunction. Also, since a causal model may not be just two layers, links of several CAUSES statements may be traversed in tracing an abnormal observation back to a component malfunction mode. In summary, the goal of abductive diagnosis is to conclude a set of component malfunctions such that the device observations logically follow from the malfunctions according to the effects captured in the causal model. Although abduction is not a legal inference like deduction, it is a useful mechanism to generate possible explanations when used in conjunction with the causal net.

2.4.2 Specific Model-Based Diagnosis Systems. The previous section explained the fundamental approaches to model-based diagnosis. There are numerous systems that perform model-based diagnosis using these core approaches, each with its own contribution to the approach it's based on. The three systems presented below are

good representatives of the consistency or abductive approach they use. All three systems are a foundation upon which other consistency or abductive systems were built. To discuss these three provides an understanding of a wide range of other consistency and abductive systems. Furthermore, the problem solving methods each system uses are contained in the set of methods for the generalized troubleshooter. Besides giving an example of how consistency and abductive systems work, discussion of them gives an idea of the types of problem solving methods that can be used in the multimethod approaches to diagnosis and ultimately in the generalized troubleshooter.

2.4.2.1 Consistency Based. The DART system is among the first model-based diagnosis systems that followed the consistency approach. It was developed at Stanford University as part of research into developing an automated diagnosis system applicable to a wide class of devices. It was implemented in MRS, a meta-level representation system, and used to diagnose problems in simple digital circuits, complex teleprocessing equipment, and the cooling system of a nuclear reactor (10:332). DART implemented the concept of using device design information as the basis for diagnostic reasoning versus the shallow knowledge in production rules. The device design information contained knowledge of a device's structure and intended behavior and these constitute the model used by the DART system.

In DART, all behavioral and structural descriptions are expressed as propositions using prefix predicate calculus. For example, in an example from (10:330), the following propositions can be used to declare knowledge about components:

To declare that components X1 and X2 are exclusive-OR gates:

(XORG X1)

(XORG X2)

To declare the behavioral knowledge of an XOR gate:

```
(IF (AND (XORG g) (VAL (IN 1 g) ON) (VAL (IN 2 g) ON))
    (VAL (OUT 1 g) OFF))
(IF (AND (XORG g) (VAL (IN 1 g) ON) (VAL (IN 2 g) OFF))
    (VAL (OUT 1 g) ON))
(IF (AND (XORG g) (VAL (IN 1 g) OFF) (VAL (IN 2 g) ON))
    (VAL (OUT 1 g) ON))
(IF (AND (XORG g) (VAL (IN 1 g) OFF) (VAL (IN 2 g) OFF))
    (VAL (OUT 1 g) OFF))
```

To declare the structural knowledge that the output of XOR gate X1 is connected to the first input of XOR gate X2:

```
(CONN (OUT 1 X1) (IN 1 X2))
```

To improve the efficiency that DART generates diagnoses, it provides the ability to state simplifying assumptions. Two common assumptions are the *single fault assumption* and the *non-intermittence assumption*. The single fault assumption states that there is at most one faulty component in a device whose failure can explain the overall misbehavior. This assumption reduces the number of suspect components to be generated. The non-intermittence assumption states that components behave consistently for the duration of the diagnosis; that is, they do not fail at time t and then behave correctly at time $t+\Delta t$.

To find a diagnosis, DART follows an iterative three-step process. In step one, DART uses the device's structural and behavioral model and statements about device observations to deduce propositions about component behavior that describe the inconsistent observations. This step produces sets of suspect components and their propositions describing inconsistent behavior. If the single-fault assumption is asserted, the suspect sets are intersected to find the singleton sets of components that are common

to all the initial suspect sets. Step two gathers additional data to help confirm or disconfirm the propositions surrounding the suspect set. DART does this by selecting a proposition from step one and deducing different inputs to the device other than the ones observed that, when used with the device model, predict certain observations. In step three, the deduced inputs are applied to the device and the new observations are input to DART, and the process repeats. If the new observations are consistent with the predictions from step two, then the proposition involving a suspect component is consistent with the device behavior it predicts, and the suspect remains a suspect. If the new observations do not agree with the predictions from step two, then the suspect component is not operating incorrectly as the proposition suggested. In this case, the suspect component is no longer suspected as operating in the assumed incorrect mode.

To produce the propositions in steps one and two, DART uses *resolution residue* (9:1), a variation of the resolution inference procedure. Resolution residue is "...a direct proof procedure rather than a refutation method" and uses the resolution rule of inference on conjunctive normal propositions (10:334). Resolution residue begins with propositions known to be true and ends when it deduces a proposition whose literals are each 1) the negation of a proposition from the device model and 2) logically consistent with all other propositions surrounding the device. To prove consistency, resolution residue attempts to prove the literal's negation. If the attempt is unsuccessful, then the literal is consistent. If the consistency test does not terminate, resolution residue fails. However, Genesereth points out that "Fortunately, the problems that arise in diagnosing most computer-hardware faults are decidable" (10:336), so the procedure is expected to be applicable the majority of the time.

A second consistency-based system for diagnosis is GDE, the General Diagnostic Engine. GDE was developed at the Xerox Palo Alto Research Center to overcome the limitations of automated diagnosis systems that preceded it. Specifically,

GDE sought to develop an efficient general method to diagnose simultaneous failures in multiple components, and provide a means for guiding the gathering of additional observations to prune the suspect component list.

The models used in GDE are based on a device's structural knowledge and knowledge of its component's correct behavior. In GDE, both types of knowledge are represented as *constraints*. Constraints are conditions that must be satisfied by the correctly operating internals of a device. Constraints can represent the connection of components, or the behavior of a component given values for its inputs or outputs. A GDE device model is a *constraint propagation net*. Given observations for the inputs to a device, structural constraints dictate the connectivity of device inputs to components, of components to components, and of components to device outputs. Behavioral constraints dictate the values at a component's terminals given values at its other terminals. For example, given values for the inputs to a two-input adder component, the correct output is constrained to be the sum of the two inputs.

To find a diagnosis, GDE starts with several assumptions: making an observation does not change the value of that observation at a later time; non-intermittence for failures, the same assumption DART uses; any behavior a component exhibits that is different from the model prediction implies the component is faulty. The last assumption is because the model only contains knowledge of correct component behavior. Next, GDE follows a three-step procedure similar to DART. In step one, the observations of a device are input to the system and the predicted values based on these observations are produced by constraint propagation. Besides the predicted values for components, GDE propagates which constraints were used to deduce those values. GDE uses an *Assumption Truth Maintenance System* (ATMS) to accomplish this. The propagated constraints are referred to by the name of the component they describe. A discrepancy occurs if a point in the constraint net has more than one value assigned to it.

This occurs, for example, if a "1" is observed at a device output terminal, yet the constraint net predicts a "0" there. In step two, GDE uses the component names carried with the predicted value to form *minimal suspect sets* of components that can explain the discrepancy. A minimal suspect set is a minimum cardinality set of suspect components that can explain the discrepancy if all the members of the set are faulty. The procedure to determine the minimal suspect sets corresponds to the general Set Covering Problem where the things being covered are the discrepancies and the things covering them are the components carried with a predicted value. As in DART, GDE requires additional data to discriminate among the suspect sets. However, unlike DART which manipulates the inputs to create new observations, GDE probes additional points within the device without changing any inputs. This is the task of step three, to determine optimal probe points. To do this, GDE computes a scalar for each point in the constraint net where there is a discrepancy that measures the expected number of additional probes required to isolate the correct diagnosis. The computation is based on the concept of minimum entropy from information theory (18:93-109). Using the domain-specific knowledge concerning the a priori probabilities of component failure, GDE calculates the scalar value. The point with the minimum scalar value represents the point with the lowest information disorder and hence the next best probe point at which to gather additional observations. The additional observations are input to the system and the cycle repeats.

2.4.2.2 Abductive Based. The AID system (Abductive and Iterative Diagnosis) is an abductive-based system designed to diagnosis problems in the hydropneumatic braking system of a truck. It was developed as part of the CHECK project, a six year project involving the Universita di Torino and Messarteam while researching automated diagnosis for industrial applications.

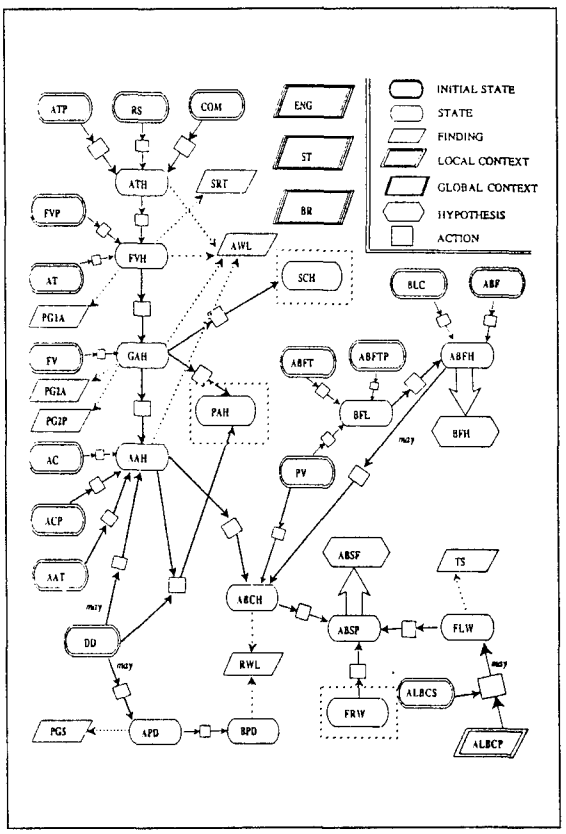
The AID system uses a causal network of the faulty behavior of the braking system. In the simple case, this means the model shows the specific ways components are

known to fail and the symptoms caused by the failures. For example, the model could show "brakes not grabbing" as a symptom caused by the specific component failure "low brake line pressure," where the component is the entire brake line system. Additionally, the model can show strings of causal links between a symptom and several component failures. For the previous example, a "hole in brake line segment X" failure can cause the failure "low brake line pressure" which in turn leads to the symptom "brakes not grabbing."

The causal network used for the model in an abductive system can be thought of in graph terms as a semantic network with specific definitions for the nodes and arcs. In AID, the nodes represent possible individual component failures, internal states the system could be in because of component failures (these states are not directly observable to the outside world), symptoms the system could exhibit, and contextual knowledge describing when the model is valid to use. The arcs are of the following types: causal arcs, that show the cause-effect relationships between pairs of failure nodes and state nodes or state nodes and symptom nodes; HAM (Has As Manifestation) arcs that link a symptom to its associated internal state nodes. Figure 2.1, taken from (4:273-274), shows a part of the causal network used in the AID system.

To determine a diagnosis using the causal network, AID performs what it calls a sequential diagnosis. The first step is to collect information about the context of the current diagnosis problem and observations of system parameters. The context information defines if a model is applicable or not. For example, in Figure 2.1, part of the model context is that the braking system is applied and that the engine is on. If an observation was that the wheels were locked, but the context of this observation was that the braking system was not applied, then the given causal model would not be correct to use under this context because the causal relationships are models of behavior when the braking system is applied. The next step is to explain, using abduction, why the observed

symptom is present. This is done by starting at the symptom node in the causal network and tracing backward via the arcs to the component failure nodes. The next step is to predict all symptoms that would be exhibited if the suspected components were actually faulty. This is done by starting from the component failure nodes and tracing forward via the arcs to the linked symptoms. This step is done to eliminate those components from suspicion that predict a symptom that is not observed. With the remaining set of components, the system asks the user to supply additional observations to further discriminate among suspects. With the new data, the cycle is repeated until the faulty component (or components) is identified and reported.



STATE

AAH	Anterior Air Hypopressure
ABCH	Anterior Brake Cylinders Hypopressure
ABFH	Anterior Braking Fluid Hypopressure
ABSP	Anterior Braking System Problems
APD	Anterior-Posterior Air Pressure Disparity
ATH	Anterior Tank Hypopressure
BFL	Braking Fluid Loss
BPD	Anterior/Posterior Braking Power Disparity
FLW	Front Left Wheel
FRW	Front Right Wheel
FVH	Four-ways Valve Hypopressure
GAH	General Air Hypopressure
PAH	Posterior Air Hypopressure
SCH	Spring-loaded Cylinder Hypopressure

INITIAL STATE

AAT	Anterior Air Tank
ABF	Air in Braking Fluid
ABFT	Anterior Braking Fluid Tank
ABFTP	Anterior Braking Fluid Tank Pipe
AC	Anterior Converter
ACP	Anterior Converter Pipe
ALBCS	Anterior Left Brake Cylinder Seized
AT	Air Tank
ATP	Air Tank Pipe
BLC	Brake Lining Clearance
COM	Compressor
DD	Duplex Distributor
PV	Four-ways Valve
FVP	Four-ways Valve Pipe
PV	Partitioning Valve
RS	Regulation System

CONTEXT

Global	BR	Braked
	ENG	Engine
	ST	Status of the truck
Local	ALBCP	Anterior Left Brake Cylinder Position

FINDING

Easy to Get	AWL	Anterior Warning Light
	RWL	Rear Wheel Locked
	SRT	System Trip Recharge
	TS	Truck Skid
Hard to Get	PG1A	Pressure Gauge 1 Anterior
	PG2A	Pressure Gauge 2 Anterior
	PG2P	Pressure Gauge 2 Posterior
	PG5	Pressure Gauge 5

HYPOTHESIS

ABSF	Anterior Braking System Fault
BFH	Braking Fluid Hypopressure

Figure 2.1 - AID Partial Causal Network

2.4.3 Model-Based Diagnosis: Advantages and Disadvantages. Model-based diagnosis has a number of advantages and disadvantages. Among the advantages are:

1. It allows a broader range of faults to be diagnosable compared to first generation systems that relied solely on precompiled lists of symptom/faults because even unexperienced faults can be diagnosed from first principles (7:23).
2. The model-based reasoning method is device independent, meaning the model-based diagnosis system can be used on a device as soon as its design information is available (7:18).
3. Model-based diagnosis encourages a disciplined approach to designing an automated diagnosis system. Models can be divided along clear lines such as structure and behavior, allowing the designer to concentrate on collecting the types of knowledge needed one division at a time (4:285).
4. Due to the modular nature described in 3, system update is easier because specific portions of the model need only be changed. This also reduces the chance of errors being introduced outside the module of interest (4:285).
5. A model-based diagnosis system provides a cleaner explanation for why a fault is present, showing what propagated constraints or causality links were involved in the hypothesis generation (4:286).
6. Component models may be stored in a model object library. Model generation for new devices using existing components is easier by reusing the models contained in the library (22:319).

Among the disadvantages are:

1. The models may become too complex. If all interactions in a device are to be included in its model, there may be no modular way to express all of them (22:311).
2. Computing all hypotheses grows exponentially with the number of components (22:311).
3. Structure and behavior are not the only knowledge applicable to troubleshooting (3:222).
4. A discrepancy between the predicted behavior and the observed behavior is not always indicative of component failure (3:223).

Although model-based diagnosis has several disadvantages, they do not prevent the model-based method from being usable. The following are ways to overcome the possible disadvantages listed above:

1. If the frequency of times is small that complex interactions contribute to a diagnosis, then the model could be built on the assumption that the complex interaction is not used and therefore is not included in the model. Additionally, the model could be a hybrid, containing a symptom/fault association for the complex interaction case.
2. Models may be organized hierarchically, with the model detail increasing as the hierarchy is descended. This scheme could allow diagnosis to the unit level, then to the card level within the unit, then to the section level on the card, then to the chip level, and so on. Each level considers a subset of the total number of components in the device. Because of this, for each level the number of possible hypotheses is less than the total number of hypotheses if considering the total number of components.
3. Human troubleshooters use other knowledge beside structural and behavioral knowledge. Chitarro and others have identified five types of knowledge

usable during diagnosis (3:228): structural and behavioral; functional knowledge, or the roles filled by components in a device; teleological knowledge, or knowledge about the goals a system or subsystem was designed to fulfill; and empirical knowledge, or knowledge obtained from experiment or experience.

4. Components may have a range of values within which is considered normal operation. The consistency-based systems previously presented rely on an equal or not-equal check for comparing predicted to observed values. Additional component fault knowledge and the context in which they operate determine if a simple equal or not-equal check is an appropriate fault discriminator.

2.4.4 Model-Based Diagnosis Conclusion. Section 2.4 has presented diagnosis as based on the model-based technique. Compared to the first generation of automated diagnosis systems, model-based systems have significant advantages that make them much more attractive. Additionally, their disadvantages are not insurmountable. Furthermore, model-based abilities contribute directly to a generalized troubleshooter where robust operation and range of applicability are important. Still, just as model-based technology advanced the state of the art over first generation systems, model-based systems themselves are a step along the evolution to more abstract and generic diagnosis systems. The systems presented above each relied on the same set sequence of methods to satisfy the goals of very similar three-step diagnosis processes: find discrepancies and determine suspects, gather additional data, and refine the suspects. What can be abstracted from model-based systems to make an even more generic diagnosis system? Since the process of the three systems was very similar, and it was realized by different methods, the proposed answer is the problem solving methods current model-based systems use can be abstracted out leading to a more generalized diagnosis system design that is not limited to

use only certain methods without choice. These diagnosis systems use a combined method approach to diagnosis.

2.5 Combined Method Approach to Diagnosis

While previous model-based diagnosis systems each fulfilled similar goals to find a diagnosis, they fulfilled the goals by using different problem solving methods. In an attempt to expand the generality of past model-based systems, the systems that use the combined method approach to diagnosis allow dynamic selection between methods based on available domain knowledge and the current state of the diagnosis process. Where a past model-based diagnosis system might fail because the problem that it is solving does not meet its knowledge requirements, a combined method system can determine what knowledge is available and dynamically select a method to satisfy the current diagnosis process goal being satisfied.

2.5.1 Meta-level Inference Systems. The first combined method approach to diagnosis is not so much an actual system, but is an architecture to organize the knowledge used in a combined method system.

The term meta means "beyond" or "about". Metaknowledge refers to knowledge about knowledge, and a meta-level inference system is one that performs inference at one level about another level. In meta-level inference systems, the "level" refers to the *object level* and the *meta level.* The object level refers to a domain-specific area of interest. Inference performed at the object level uses domain-specific knowledge to deduce results within that domain. The meta-level, on the other hand, refers to a level above the domain level. The meta level contains knowledge about the knowledge in the object level.

Another important division in meta-level inference systems is distinguishing between domain knowledge and control knowledge. Domain knowledge, mentioned

above, refers to that which is known about a specific domain. It embodies objects, relations, facts, rules, and procedures of the domain of expertise (13:3). Control knowledge is knowledge about how to use the domain knowledge.

Meta-level inference systems are useful for many things (6:1), but for a combined method approach to diagnosis, they are useful to control the inferencing in the object level. Used this way, the meta-level contains control knowledge about how to use the domain knowledge, contained in the object level, for a specific purpose. In terms of the combined method approach to diagnosis, the domain level contains the problem solving methods that are designed to perform on a narrow range of domain assumptions and knowledge requirements. The meta-level contains knowledge about the overall diagnosis process and when to invoke a problem solving method based on the state of the diagnosis process and presence of any device domain knowledge.

2.5.2 Task Structures. Given an architecture to organize the knowledge used in combined method diagnosis, the next step is to state a common definition of the diagnosis process and determine a set of methods useful during that process.

The task structure approach to combined method diagnosis begins by decomposing diagnosis into its constituent tasks to be performed. Each task has a goal to be achieved and specifies *what* is to be done. A task is characterized by its *knowledge roles*, or the type of inputs and outputs it receives and produces. It is further characterized by the problem solving methods useful to achieve its goal. Problem solving methods define *how* the goal of a task is to be accomplished. Problem solving methods also have knowledge roles, and may be further decomposed into the tasks that comprise it. This recursive decomposition ends when a *primitive inference* task is reached. A primitive inference task is one where inference is performed using domain knowledge to attain a specific goal. The decomposition of diagnosis into its constituents in this manner forms a *task-method* graph.

Given the knowledge roles for problem solving methods, a system using the task structure can reason about which method to choose from the multiple methods associated with a task. *Suitability criteria* state method-specific knowledge that entails the knowledge roles and reflects the domain-specific knowledge requirements and assumptions necessary for a problem solving method to operate. Given the domain-specific knowledge and current diagnosis task, control knowledge can dynamically determine which problem solving method is most appropriate to apply by evaluating each method's suitability criteria.

2.5.3 ARTIST. ARTIST was a 32 month project aimed at developing model-based reasoning methods for diagnosing industrial systems. It involved a consortium of European universities and industries and ran from 1990 to 1993. Its goal was to develop a generic architecture for a model based diagnosis system that could have previous model-based systems as instantiations of its architecture. This goal was important to realize to the ARTIST team because in industrial operations, where equipment changes are frequent, updating first-generation diagnosis system knowledge bases takes too much time. Furthermore, the team realized the reuse potential of existing model-based diagnosis systems and sought to develop an architecture to reuse those systems.

The main contributions of ARTIST are the *ARTIST architecture* and its *specification methodology*. The architecture divides the diagnosis process into three modules: the Diagnostic Strategist, the Predictor, and the Candidate Proposer. The Diagnostic Strategist controls the overall diagnosis process by evaluating diagnosis goal accomplishment, determining which device models or parts thereof should be focused on, specifying the criteria suspect components needs to satisfy to be further investigated, and determining the next diagnostic step. The Predictor produces predicted component values based on the device inputs and detects discrepancies based on the device observations. The Candidate Proposer defines suspect components based on the discrepancies, ranks the

suspects according to the criteria specified by the Diagnostic Strategist, and accepts new observations to discriminate between suspect components. The specification methodology provides a mapping from the characteristics of a diagnosis problem to the various model-based diagnosis systems available to ARTIST that can work on those characteristics. The methodology identified three areas of diagnostic problem requirements: task requirements (such as limitation of resources), fault requirements (such as the single fault assumption), and model requirements (such as static versus continuous device model). It also identified the properties of the three model-based diagnosis systems it used for reuse in terms of the diagnostic strategy specification, the predictor specification, and the candidate proposer specification. These specifications identify the knowledge and assumptions of the three model-based diagnosis systems. The methodology then links the problem requirements to the system specifications, and the ARTIST architecture is then instantiated with the combinations of methods from the model-based diagnosis systems.

ARTIST does not perform dynamic method selection during diagnosis. Once the method selection has been established by the methodology, it remains fixed for the duration of the diagnosis session. With a new diagnosis problem, however, new methods may be chosen based on the evaluation of the new problem's requirements and how they match to the method properties.

2.5.4 SRS. The last combined method approach to diagnosis actually involves more than combining problem solving methods. It combines four reasoning paradigms to opportunistically use the strengths of a paradigm, when appropriate, under the control of a central algorithm. The system is called SRS for Synergistic Reasoning System.

The SRS uses case-based, rule-based, model-based and procedural reasoning together. The SRS prototype was built to diagnose problems with the Reaction Wheel Assembly in the Hubble Space Telescope. The SRS uses a central control algorithm and a number of heuristics to activate different reasoning paradigm modules when appropriate

during the diagnosis process. Also, it uses a modified blackboard problem-solving model to organize the reasoning paradigms, heuristics, control scheme, and solutions. The central control algorithm cycles through four stages: cooperation, confirmation, refutation, and follow-up. During cooperation, different reasoning modules post partial solutions to the blackboard. Confirmation allows other reasoning modules to verify the results posted to the blackboard by comparing their partial solution to what was posted. This is done to increase confidence in the partial solutions being posted. Refutation allows other reasoning modules to refute the results posted to the blackboard similar to how confirmation verifies partial solutions. Finally, follow-up examines the history of diagnoses to determine if any problematic trends are indicated. For example, if the same diagnosis and repair plan was reached many times in the near past, follow-up would suggest the presence of the deeper problem requiring its own diagnosis. The heuristics provide guidance for reasoning module selection or diagnosis goal persuit. For example, one heuristic is "If more than one reasoner can act on a goal, employ in the order of rule-based, case-based, model-based". Another heuristic is "If multiple components are suspected, diagnose the least reliable." The combined effect of different reasoning modules, the control algorithm, and heuristics is that a diagnosis problem can be solved through cooperative efforts that no one reasoning module could solve on it own.

2.6 Literature Review Conclusion

This chapter has presented a brief summary of past efforts for automated diagnosis. Since the first generation diagnosis systems were constrained in their ability to contribute much to a generalized troubleshooter, an alternate diagnosis reasoning method was presented. The model-based method overcomes the narrow applicability of first generation systems and provides a significant step forward in generalized diagnosis system

design. Analysis of several model-based systems led to the idea that even they could be generalized by considering the problem solving methods implicitly encoded in them. This led to considering diagnosis systems based on the ability to choose among several problem solving methods using meta level reasoning as a further step toward a generalized troubleshooter.

III. Methodology

3.1 Chapter Overview

This chapter discusses the high level methodology used to construct a generalized troubleshooting architecture. It discusses the scope this work focused on and provides reasons for the focusing assumptions. Also, it presents the relation between the work performed and the research objectives to demonstrate how the objectives were satisfied. Chapter two presented past efforts in accomplishing automated diagnosis. The approach for this work is to combine the strengths of those past efforts in constructing a generalized troubleshooting architecture.

3.2 Scope

This thesis deals with performing troubleshooting, the process of determining why an entity is not working properly and implementing corrective actions that return the entity to correct operation. The results of this thesis form an initial architectural basis for realizing the vision sought by a generalized troubleshooting system. The vision is an unconstrained statement of what is desired of a generalized troubleshooter regardless of current technology. It provides a long range target for what a generalized troubleshooter eventually ought to be. The vision's main characteristic is to have an automated system capable of successfully troubleshooting a broad range of different domains. Regardless of the domain, a truly generalized troubleshooter could determine why problems existed by drawing on and applying "whatever is available." This statement is intentionally vague because at present, all that encompasses "whatever is available" is not defined. Furthermore, even when there is no obvious step to take, a generalized troubleshooter

would use "whatever is available" to at least make attempts at solving the problem. Because the long term vision of a generalized troubleshooter is so broad, it needs to be narrowed so that gains can be made in the short term. One way to narrow the vision is to limit the range of domains. This limited range could include dealing with only physical entities like electronic devices, a truck's braking system, or a nuclear power plant, or abstract entities like management or process control. A further restriction is to limit how the entities in the limited range are represented. For this thesis, this restriction meant dealing only with entities capable of being represented by interconnections of subsystems. Since even these restrictions lead to a broad problem space, I made further focusing assumptions to obtain a workable problem size. These assumptions further limited what was considered regarding the applicable domains, the troubleshooting process itself, and the degree of "whatever was available." The last item refers to limiting the vast array of what *could* be available to truly generalized troubleshooter down to considering only one type of reasoning and the domain knowledge it depends on. The three focusing assumptions were:

1. Focus on the first sub-goal of troubleshooting: determine why an entity is not operating properly;
2. Focus the entities undergoing troubleshooting to physical devices;
3. Focus on using model-based problem solving methods to perform the reasoning that generates hypotheses for why the device is failing.

Also, a particular approach to diagnosis was adopted. That approach involves combining problem solving methods, each with their own strengths and knowledge requirements, to work together on solving a diagnostic problem. Additionally, using the methods should be opportunistic. That is, according to the current state of the diagnosis process, dynamically choose among the problem solving methods to use the one that can contribute the most.

Focus one deals with performing diagnosis, perhaps "...the single largest category of expert systems in use" (11:xi). Diagnostic expert systems attempt to automate the

diagnostic reasoning process for various domains including medicine, electronic devices, and plant process control. Diagnostic expert systems have been under investigation since the mid 1970's when the medical advice system MYCIN was developed for diagnosing blood infections (24:642). Besides providing a basis for research into basic artificial intelligence technologies, diagnostic expert systems have the potential for wide practical use. For example, a goal of the ARTIST program was to design an environment that made it relatively easy to build diagnostic systems for realistic industry applications such as diagnosing problems in power generation and transmission systems, operation and supervisory systems in refineries, and systems for chemical, food and cement manufacturing (21:9). The diagnosis part of the troubleshooting process was examined because of the abundant research conducted in automated diagnosis and the desire to use the generalized troubleshooting architecture in a substantial application with an automated diagnosis need.

The application that this research could be used in is the MAGIC (Multimission Advanced Ground Intelligent Control) program. The program is managed by the Satellite Control and Simulation Division of Phillips Laboratory at Kirtland Air Force Base. It is a program to improve satellite control by using an intelligent real-time system to "manage and control multiple satellite constellations, easily adapt to new constellations, improve operator effectiveness and enhance operational capabilities" (14:7). Due to a change in the operational concept of satellite operations, highlighted by a reduction in the training and rank of satellite operators and the elimination of an engineering shop to troubleshoot anomalies not experienced before, an "efficient and economical automated system is necessary to assist the current satellite operator in...maintaining...high priority DOD..." satellites (14:1-7). A generalized troubleshooter would assist a program such as MAGIC by being able to automatically diagnose (possibly unforeseen) problems on many satellites in a time when the job for a human expert to do this has been eliminated.

Focus two, focusing the entities undergoing troubleshooting to physical devices, stems in part from MAGIC. Also contributing to this focusing assumption is the abundance of research conducted in developing diagnosis systems for devices. The abundant research results of previous diagnosis systems provided a large base from which to choose ideas.

Finally, focus three deals with using a specific kind of method to generate hypotheses for device failure. The method and its knowledge should support several characteristics. Among these are: be general enough to be applicable to various devices without significant rework; use knowledge not based solely on experience so that upgrades to the diagnosis system are easy; organize knowledge cleanly; recreate the reasoning process, not just use recorded results; and provide a clear explanation for why a hypothesis may cause a fault. Also, the method should support the needs of the troubleshooter envisioned in the MAGIC system. Model-based diagnosis satisfies these characteristics. The basic paradigm of model-based diagnosis, comparing observations to predictions based on a model of the device, works well to be device independent. Furthermore, updating a device with new or different components does not require a lengthy experience buildup with the new device; rather, the model can be updated or reused from a model library and then used with the previous reasoning method. Also, the knowledge used in model-based diagnosis is divided among clean lines such as along component lines, along structural lines, or along functional lines. Cleanly organizing diagnostic knowledge leads to a principled approach to designing a diagnosis system and reduces missed knowledge by exposing gaps in model content. Finally, model-based diagnosis uses knowledge a human troubleshooter uses to reason from first principles about device behavior, again reducing the reliance on experience and providing the means to diagnose unforeseen problems.

These focusing assumptions allowed the scope of work to be reduced to a manageable size. However, they also imposed some limitations on the generalized

troubleshooting architecture built using them. Because the architecture focuses on performing diagnosis, the second half of the overall troubleshooting process is ignored. Once the cause of a fault is found, it needs to be repaired to return the device to operation. The generalized troubleshooting architecture ignores for the moment the issue of how to correct the fault once its cause is found. Additionally, the architecture relies on model-based methods that themselves rely on models. If models do not exist, then model-based methods cannot be used. For the target application, where satellite subsystems would undergo diagnosis, this limitation is not too severe because ample design data exists for satellite components. It is the design data from which models are built, so the satellite domain would more than likely have the models usable by the model-based methods. Chapter five, Analysis, presents ways these focusing assumptions may be relaxed and consequent ways the generalized troubleshooting architecture can be expanded.

Besides the focusing assumptions listed above, another factor played into the methodology for this work. That factor involved not trying to discover the one best algorithm a generalized troubleshooter should use, but to build a troubleshooter using a toolbox approach. Just as a toolbox contains different items with which to do work (claw hammer, pliers, ratchet wrench, crescent wrench, etc.), the generalized troubleshooter has different tools available to it to do diagnosis. Each tool is designed separately to do a specific job well. Designing a tool for a narrow range of use is often easier than designing a tool that is useful for everything. Concentrating on a narrow range of use can allow the tool to be designed for efficient use in that range. Then, if the union of narrow ranges is broad enough, the combined use of the different tools can lead to the accomplishment of work that no tool on its own could do. This approach to problem solving, a systems approach, is widespread and applying this to troubleshooting appears to be a valid. In hardware design, the current trend is to design new devices by choosing from existing off the shelf components specialized for a certain function. Depending on the requirements of the new device (for example, size or speed), components able to contribute to the new

device requirements are chosen. Likewise, a very different problem solving discipline, program management, also uses a systems approach. Here, the current trend is to use Integrated Product Teams (IPTs) composed of personnel who are specialists in different areas (finance, contracting, test, configuration management, systems engineering) all under the control of a program manager. The program manager assigns team members to tasks based on the work to be accomplished. A third area, software engineering, has a vision of using the systems approach to be able to choose among existing specialized code when building new software systems. Although software engineering has not had the degree of success with the systems approach compared to the hardware and program management areas, the area is actively designing systems able to realize the systems approach vision. A generalized troubleshooter using the systems approach operates in a similar manner to these three examples. Under the control of a central module, it chooses from existing specialized methods to accomplish diagnosis tasks. Given that the systems approach is being applied to a wide range of areas, applying it to the problem of troubleshooting appears to make sense.

Depending on the problem at hand, a mechanic or a generalized troubleshooter ought to choose the most appropriate tool (based on some parameters) to do the work at hand. The parameters evaluate the data and knowledge available at the time of tool selection (25:2). Also, if after work on the problem started, new knowledge came about that showed the current tool was not doing the job, a mechanic or generalized troubleshooter ought to be able to change to another appropriate tool based on the new situation. If, for example, the (mechanical) job at hand was to tighten a nut onto a bolt, and the speed it was put on was the discriminating parameter, the first choice of tool from those just listed would most likely be the ratchet wrench. But what if after tightening the nut, enough of the bolt shaft was left so that the socket on the ratchet could not reach the nut anymore? The mechanic would need to reevaluate tool selection and make another appropriate choice. In this example, the choice may be to go to the crescent wrench. The

availability of different tools allows decisions to be made about the most appropriate one to use based on the problem input and allows for continued effort toward solving a problem if one tool fails.

Another reason for using multiple tools is because the goal of problem solving can be different from problem to problem. Another mechanical example could involve driving a nail. Suppose the job was to drive a finishing nail. Usually, a goal associated with driving a finishing nail is to not mar the wood. Focusing on tool selection, appropriate tools could be a claw hammer and a punch set: the hammer to drive the nail to within 1/8 inch of the wood surface and the hammer and set to drive the head of the nail flush with the wood surface. Driving a finishing nail has a different goal compared to driving a large 8 penny nail into roof sheathing. The neatness goal does not count as much here, so using the claw hammer itself would be appropriate.

These mechanical examples have a relation to diagnosis. Determining hypotheses for fault causes is analogous to performing a mechanical job. Both are solving some problem. The tools the mechanic uses are analogous to the problem solving methods the generalized troubleshooter uses. Furthermore, the mechanic can dynamically evaluate tool selection based on how the tools are contributing to solving his problem. Likewise, the generalized troubleshooter can dynamically evaluate what problem solving method to use based on the current state of the diagnosis process. In both circumstances, different tools are used together to solve a problem. Also, tool selection is "...based on given input (goal and specific problem), the available knowledge of the problem type and...problem solving methods" (25:2). Of the tools available, the most appropriate is used when its strengths are needed.

3.3 Research Methodology

This section discusses how the work for this effort was organized and reviews the development of the research objectives. Also, it discusses in more detail the phases of work mentioned in Chapter one, section 5, and how they relate to the research objectives.

3.3.1 Research Objectives. The research objectives for this thesis aimed to The overall objective of this research was to investigate the following hypothesis: a generalized troubleshooting system can be designed which uses general troubleshooting heuristics and domain knowledge if available to guide the troubleshooting process. In light of the combined problem solving method concept and focusing assumption one mentioned above, the overall objective was redefined to be: determine a troubleshooting system architecture that can dynamically choose between competing problem solving methods when performing diagnosis. To assist in managing the research, the overall objective was decomposed into four sub-objectives. This helped to guide the work and provide a way to measure progress in achieving the overall objective. These sub-objectives are (see also Table 3.1):

1. Determine a set of problem solving methods in terms of domain and device independence (Research Objective 1.1);
2. Determine a set of the knowledge the problem solving methods require (Research Objective 1.2);
3. Determine a basis for which heuristics can be developed to choose between problem solving methods (Research Objective 1.3);
4. Determine an organization into which the problem solving methods can be placed (Research Objective 1.4).

Satisfying the four sub-objectives satisfies the overall objective. That is, to have an architecture that combines problem solving methods, first those problem solving methods need to be identified. This is sub-objective one. Next, a particular stage a diagnosis

solution is in needs to be matched to an appropriate problem solving method to advance the solution. This requires evaluating diagnosis process parameters in terms of a method's specification to evaluate a method's applicability. The specification contains what knowledge the method requires in order to operate, what kind of diagnosis goal it is designed to efficiently satisfy, and what type of result it is designed to produce to name a few. Sub-objectives two and three are intended to cover these points, namely defining the specification knowledge for problem solving methods and providing a basis for heuristics to choose which problem solving method to use after the parameter evaluation. Last, sub-objective four provides the framework into which the other three objects can be placed and, for this work, provides the initial design.

Because this work's aim was to build on the strengths of previous combined method approaches to diagnosis, a second level sub-objective was created to accomplish this: Determine complementary strengths (in terms of organization and selection) of previous combined method approaches (Research Objective 2.1). Its accomplishment was necessary to satisfy sub-objectives three and four. The primary strengths it examined were strengths related to system architecture organization and method evaluation and selection.

3.3.2 Research Organization. The objectives laid out above can be visualized graphically as follows:

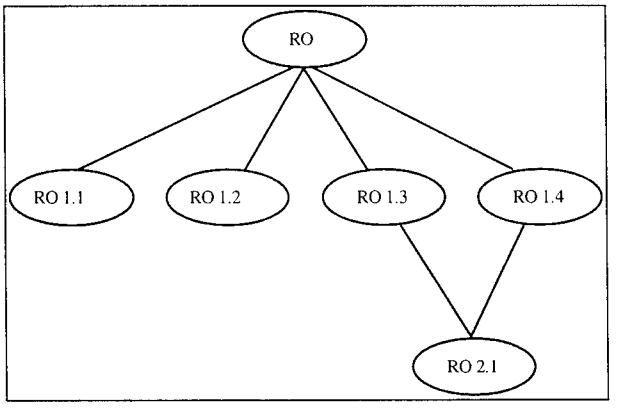


Figure 3.1 - Research Objective Organization

The nodes represent research objectives or sub-objectives and the numbers inside the nodes represent objective identification numbers (see Table 3.1). Nodes with multiple sub-objectives need all the sub-objectives satisfied for itself to be satisfied. Visualizing the objectives in this way helped to guide the research by showing how a goal could be satisfied by tracing down the graph, and by showing why a sub-objective was being investigated by tracing up the graph. Although simple, focusing on the goal structure helped to keep the research from wandering too far from where it was needed.

In order to satisfy the research objectives, work packages were assigned to them. These are the phases mentioned in Chapter one, section 5. The objectives and work packages together constitute the entire research effort organization. They can be visualized graphically as in Figure 3.2. Additionally, the research objectives and work packages together are summarized in Table 3.1:

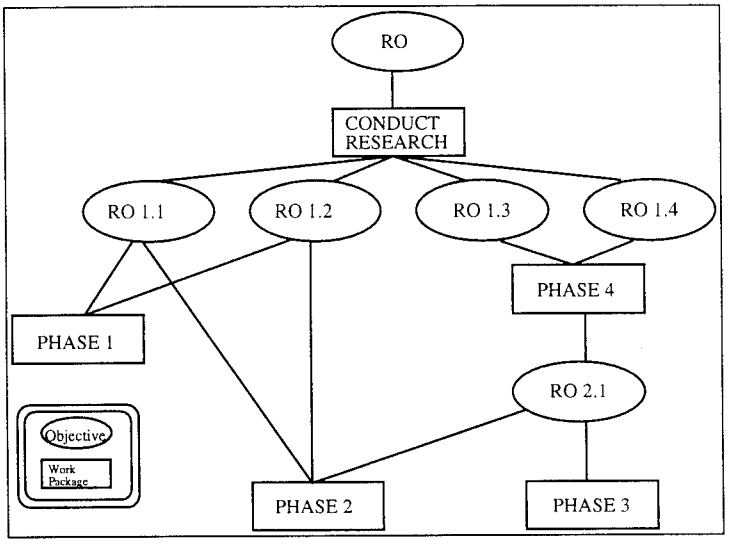


Figure 3.2 - Research Objective/Work Package Organization

Phase one consisted of learning more about diagnosis itself: the processes involved, the goals to be satisfied, and the tasks required to meet the goals. Also, after research into diagnosis itself, the model-based systems to do diagnosis were examined

Designator	Description
Objectives	
Overall Research Objective (RO)	Determine a troubleshooting system architecture that can dynamically choose between competing problem solving methods when performing diagnosis.
RO 1.1	Determine a robust set of problem solving methods in terms of domain and device independence.
RO 1.2	Determine the knowledge the problem solving methods require.
RO 1.3	Determine a basis for which heuristics can be developed to choose between problem solving methods.
RO 1.4	Determine an organization into which the problem solving methods can be placed.
RO 2.1	Determine complementary strengths (in terms of organization and selection) of previous combined method approaches.
Work Packages	
Phase 1	Research diagnosis and various model based systems to do diagnosis.
Phase 2	Research into other work involving a combined method approach to diagnosis.
Phase 3	Analyze four specific combined method approaches in order to find strengths from each (in terms of organization and selection) that are complementary with other approaches strengths.
Phase 4	Combine the strengths from phase 3 to form a generalized model based troubleshooter.

Table 3.1 - Research Objectives and Work Packages

since it was the model based paradigm that was the focus for this research. Phase one contributed to research sub-objectives one and two. It examined specific model-based diagnosis systems in order to find what among them could be abstracted to a higher level where a generalized troubleshooter was envisioned to operate. Looking at the model based diagnosis systems individually to extract out their components proved to be difficult. First, I did not know what components I was looking for. Although it turned out the approach this research took involved *problem solving method* components, I was unaware of this at the start of phase one. Often, the methods were not explicitly identified, nor were the knowledge requirements for the system. Only after comparing many systems were a few commonalties observed between them in terms of methods and knowledge they used. Additionally, assumptions each system made concerning, for example, the goals of diagnosis it was satisfying were seldom explicitly stated, especially in the older systems. Assumptions each diagnosis system were working under are also knowledge requirements that can be used during method selection and therefore were needed to be extracted as part of the knowledge identification. Last, most of the model-based diagnosis

systems touted themselves as being domain independent. Since a generalized troubleshooter seeks to be applicable to many domains, at first it seemed these systems fit the generalized troubleshooter description. It took time to realize that even though they were each domain independent as far as devices are concerned, the domain of the methods they used was fairly narrow. It is this aspect of domain independence a generalized troubleshooter aims to expand also. For these reasons, this phase proceeded slowly. It required understanding a wide range of systems in order to be able to classify their components.

In light of the difficulties from phase one, and the realization that each system had its own specialized set of methods whose basic purposes were shared between systems, phase two started to see if other research had extracted the commonalties between various model based diagnosis systems. In hindsight, had I realized earlier that many methods were in use, the focus of phase one could have been narrowed to search for knowledge requirements only instead of searching for both a taxonomy of methods and their requirements. Phase one would have then most likely taken much less time. Phase two was fruitful with results in that several approaches were found that each examined a piece of the problem solving method taxonomy. However, finding these approaches again was time consuming since the sense from the phase one research was that making a generalized troubleshooter was the next step for automated diagnosis. In fact, compared to phase one's literature, phase two's literature was generally newer and less abundant. Phase two supported research sub-objectives one and two and also supported the second level sub-objective to determine complementary strengths of previous approaches by providing a set of approaches to examine.

Phases three and four were the work packages that dealt with devising the generalized troubleshooter itself. Of the approaches found in phase two, four were found to have concepts applicable to a generalized troubleshooter. Phase three examined the strengths of these concepts in terms of how method selection and method organization

were done to find which ones could be merged. Phase three centered on ideas of *meta level control* (knowledge about when to use knowledge associated with problem solving methods) and the concepts that dealt with this were the ones examined. Phase four was the building package where the pieces gathered from the research were finally put together.

Due to the length of phases one and two, the essence of the ideas for this thesis was not generated until late in the research process. Although it was initially a goal to implement a generalized troubleshooter to compare its performance to the individual diagnosis systems advertised as being general, enough time was not available to make a proper implementation. A proper implementation would involve coding and testing a set of problem solving methods greater than the number used in a system examined in phase one, coding the meta-level control structure to implement dynamic method selection, developing device models, and obtaining the phase one diagnosis systems or implementing them to work as they were described in the literature so results could be compared. Instead of implementation, this thesis presents an initial design on which to base a future implementation .

3.4 Methodology Conclusion

This chapter provided an overview of the methodology used to conduct this research. It discussed choices for what was done and provided reasons for why those choices were made. Its aim was to show the reasoning process with justifications for how the research was organized. Furthermore, it demonstrated the wide range of knowledge investigated in order to satisfy the overall research objective.

IV. Design

4.1 Chapter Overview

This chapter presents the product of the work done for this thesis, an architecture for the diagnosis task of the generalized troubleshooting system (GTS). It describes the assembly of the many pieces that were important to the development of the GTS. It links the architecture functional components of the GTS spawned by the research objectives to the architecture structural components that evolved from this research. Furthermore, it fills in the structural components with the concepts produced from phases two and three. First, the GTS big picture is discussed. Next, the architecture is developed according to its three structural components: the method section, the knowledge section, and the control section. The chapter ends with a review of the completed architecture and an example of its operation.

4.2 Generalized Troubleshooter Architecture

4.2.1 Architecture Overview. The architecture for the GTS aims to concisely organize the broad scope of knowledge used in the multiple method approach to diagnosis. It also aims to provide an environment where the many problem solving methods can work together to solve a diagnosis problem, each contributing to the process when it is able and appropriate. However, although GTS stands for *Generalized* Troubleshooting System to convey the idea that GTS can be used on many types of systems, GTS is not generalized enough to be used on *any* type of system. Specifically, GTS is applicable to those systems that can be modeled by interconnections of subsystems. *Generalized* still means that GTS can be used on many systems; however the

types of systems are restricted. However, since engineered fit this interconnection of subsystems characterization, the modeling restriction does not significantly reduce the types of devices GTS was designed to diagnose.

Work phases two and three found specific concepts and an existing artificial intelligence technology, the blackboard problem solving model, to contribute to the architecture. The concepts and blackboard problem solving model were combined in phase four to make up the overall generalized troubleshooter architecture. At a high level, the overall GTS architecture covers the following items (see Figure 4.1): the *architecture functional components* (obtained from the research objectives) of methods, organization of methods and knowledge, and dynamic selection of methods during the diagnosis process; the *architecture structural components* (obtained form the thesis research results) of the method section, the knowledge section, and the control section; the *links* between the functional components and the structural components; and the *concepts* from past automated diagnosis research that are reused in the GTS' structural components. The mix is a complex one, and in an attempt to present it as clearly as possible, the GTS architecture will be presented according to the structural components.

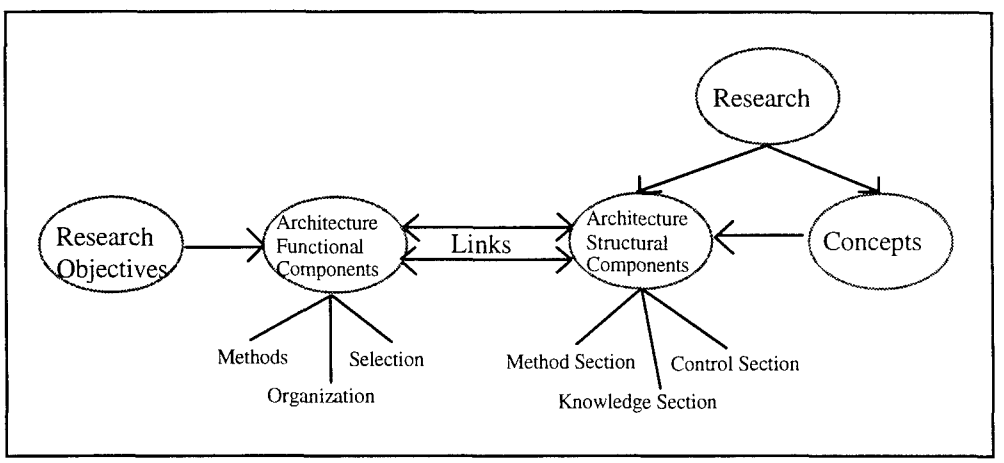


Figure 4.1 - GTS Architecture Big Picture

4.2.2 Architecture Structural Components. As the research evolved, three structural components emerged as broad categories of the GTS architecture. These are contrasted with what I call the functional components of the GTS that organize the functions the GTS architecture ought to support. The first structural component is the method section of the architecture which holds the actual model-based problem solving methods obtained from the previous automated diagnosis systems. The second is the knowledge section that holds the knowledge about the state of the diagnosis process, the domain specific knowledge particular to a diagnosis problem, and the knowledge about the applicability of the problem solving methods. The last section is the control section that holds the items necessary to dynamically choose among the methods in the method section.

4.2.2.1 Origin of Architecture Structural Components. This section describes which research concepts led to the design of the structural components for the GTS architecture. The concepts directly correspond to the main structural components of the GTS, and the structural components derived from them provide a sound foundation to organize other research concepts that were the strengths of other automated diagnosis systems. It is these strengths that fill in the foundation the structural components provide. Figure 4.2 and Table 4.1 summarize the discussion presented in this section.

The three structural components are the result from two main areas of research. The first area involves using meta-level inference systems to control a reasoning process. As discussed in Chapter two, section 5.1, meta-level inference systems rely on the separation of control and domain knowledge, and the explicit representation of control knowledge (13:25). Domain knowledge usually concerns the *what* that a system knows and control knowledge refers to *how* to use the domain knowledge. The explicit representation of control knowledge allows the strategy of how to apply domain knowledge to be defined independently from the development of the domain knowledge itself. These two ideas correspond with the objectives of the GTS. A multimethod

diagnosis system relies on both domain and control knowledge. The "what" knowledge corresponds to the knowledge the system has about a particular device under diagnosis, the knowledge it has about the diagnosis process (the diagnosis domain), the knowledge it has about the current state of the diagnosis process at a particular time, and the problem solving methods themselves. The "how" knowledge corresponds to the knowledge about when to apply the methods and to a lesser extent the steps performed by the problem solving methods during their operation. Additionally, the "what" corresponds to the object level and the "how" corresponds to the meta level as discussed in Chapter two, section 5.1. An important note is that the "how" knowledge for the steps performed by the problem solving methods is at the object level since it is particular to the methods on the object level. Furthermore, separating the different types of knowledge goes along with the toolbox approach adopted by the GTS. The problem solving methods can be designed to efficiently solve certain classes of problems independent of the control knowledge designed to efficiently use the methods. The meta-level inference idea of separating types of knowledge gave hints to the need for all three of the structural components. The domain knowledge goes into the knowledge and method sections, and the control knowledge goes into the control section.

The second research area that helped define the structural components is the blackboard problem solving model. It has many characteristics that relate to the objectives of the GTS and the ideas from meta-level inferencing, making the blackboard organization a natural foundation for the GTS organization.

The basic blackboard problem solving model involves a *blackboard data structure* that is common to a number of *knowledge sources* with the operation of the system guided by a *control module*. A blackboard model has several things to offer that are applicable to the GTS:

1. The blackboard data structure is the global database and holds the domain knowledge needed by the knowledge sources and the results of applying the knowledge

sources. Furthermore, the blackboard is usually divided into "any type of hierarchy appropriate for solving the problem" (8:4). The GTS requires that different methods operate on common information or on the results of other methods. Having the information and results globally stored for access by the various methods can be accomplished by using a blackboard data structure. Also, some methods operate only on certain information and do not need to be concerned with the information for other methods. Dividing the blackboard so that methods need to only look in the area they are interested in is an appropriate hierarchy for multimethod diagnosis problem solving. The blackboard goes into the knowledge section structural component.

2. The knowledge sources are separate and independent entities that each operate in their own special domain. The knowledge sources communicate only with the blackboard and hence only need to be aware of the portion of the blackboard that pertains to them. The knowledge sources are responsible for evaluating the blackboard contents to determine if it contains what they need to operate. Given these jobs to perform, the knowledge sources are broken into two parts: the applicability evaluation part and the body or operational part. The knowledge sources directly relate to the various problem solving methods the GTS uses. Each problem solving method is designed to work on its own special class of problems and is separate from other methods even if the methods can do the same job. A main aspect of the multimethod approach in GTS is to know when to apply a method. This relates to a knowledge source's job of evaluating its applicability. A difference between GTS and the blackboard model is that in GTS, the knowledge sources do not evaluate if they are applicable, but only contain the knowledge saying what their applicability requirements are. The applicability evaluation part of the knowledge sources goes into the knowledge structural component and the body goes into the method structural component.

3. In a blackboard system, the control module determines which knowledge source to activate based on the current solution state and how well a knowledge source

can contribute to advancing the solution. This directly relates to the requirement within GTS to choose the most applicable problem solving method at the time to advance the diagnosis process. The function of the blackboard control module led to the development of a control section for the GTS architecture to perform a similar function. The control section uses the knowledge in the applicability evaluation part of the method knowledge sources to decide which method is most applicable.

In summary, the ideas from research into meta-level inference systems and the blackboard problem solving model led to the development of three structural components for the GTS architecture. These components are shown in Figure 4.2. Also, Table 4.1 summarizes which concepts contributed to the creation of the components. The operational concepts from these two research areas directly related to the functions the GTS architecture ought to support. Also, the organizations described in these two research areas naturally provided a structure that could be used to organize the remaining concepts used in the GTS. The following three sections describe the content of the structural components and the design of how they interrelate.

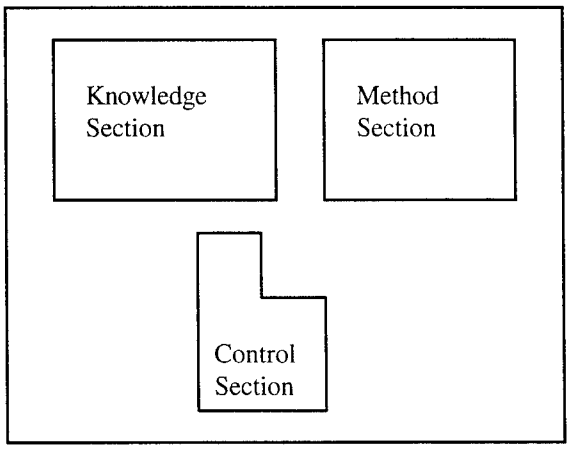


Figure 4.2 - GTS Structural Components

Structural Component	Derived from	Contains These Research Concepts
Method Section	Meta-Level Inference Systems. Blackboard Problem Solving Model.	Domain Knowledge. Knowledge Source Independence. Knowledge Source Body.
Knowledge Section	Meta-Level Inference Systems. Blackboard Problem Solving Model.	Domain Knowledge. Blackboard Data Structure. Blackboard Hierarchy Divisions. Knowledge Source Applicability Evaluation.
Control Section	Meta-Level Inference Systems. Blackboard Problem Solving Model.	Control Knowledge. Blackboard Control Module.

Table 4.1 - Structural Components: Origins and Contents

4.2.2.2 Method Section. A major point of this research was to investigate how to create a GTS architecture using the toolbox approach discussed in Chapter three. The tools are items designed to perform well on a narrow range of tasks. In terms of diagnosis, the tools correspond to different problem solving methods. These methods are designed to satisfy the goals of the tasks to be performed during diagnosis. The tasks specify what is to be done, and the problem solving methods specify steps the method takes to transform inputs to outputs in order to satisfy the prescribed diagnosis goal. Depending on the steps a method implements, different granularities of results may be produced, different knowledge may be needed to make the method work, or different processing times may be required. It is the methods that are dynamically chosen in GTS.

The method section is the place where the problem solving method code bodies are placed within the GTS architecture. A method constitutes the steps it was designed to carry out and the control mechanism that operates the steps. The method section is divided according to the concept of the knowledge source from the blackboard model. Each method has its own area in which it resides. Since the areas are separate, each method is free to be designed however is best for it to work on its particular class of problems. A result of this is the methods do not all have to be in the same representation.

If an algorithm is a suitable representation for one method, it can be placed that way in its own partition independent of the IF-THEN rule structure, for example, that may be suitable for another method. The methods do, however, need to be able to communicate in a common way with the portion of the knowledge section it shares with other methods. The expanded method section is shown is Figure 4.3. Table 4.2 summarizes the Method section contents.

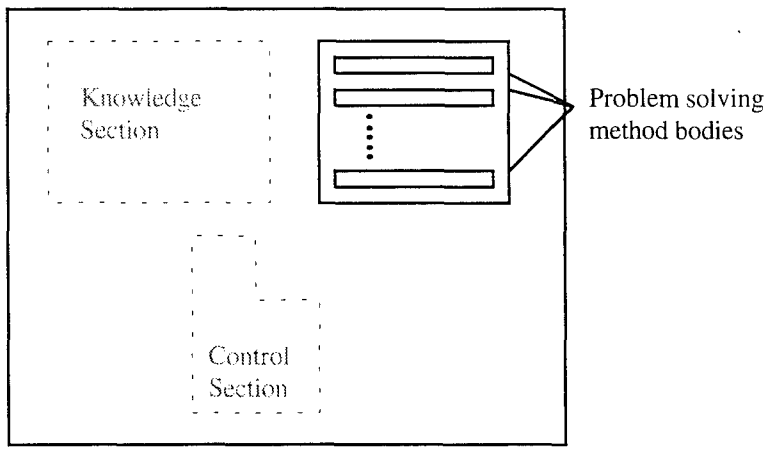


Figure 4.3- GTS Architecture, Expanded Method Section

Structural Component	Contains These Research Concepts	Filled by this Previous-System Strength
Method Section	Domain Knowledge. Knowledge Source Independence. Knowledge Source Body.	Actual Problem Solving Method Bodies.
Knowledge Section	Domain Knowledge. Blackboard Data Structure. Blackboard Hierarchy Divisions. Knowledge Source Applicability Evaluation.	
Control Section	Control Knowledge. Blackboard Control Module.	

Table 4.2 - Method Section Contents

Since this thesis focuses on the architecture of the GTS, a discussion of the internal method operations is outside the scope of this work. The overall set of methods used for the GTS and a brief description of their operation is, however, contained in Appendix A.

4.2.2.3 Knowledge Section. In designing the GTS architecture, a wide array of knowledge came into play. The knowledge section's job is to apply some structure to the diversity of knowledge applicable for use in the GTS. The knowledge included knowledge of a specific device (its component list, its models, failure data, etc.), diagnosis process knowledge (what diagnosis goals need to be accomplished, the tasks to be performed during diagnosis, and the incremental solution leading to a diagnosis), knowledge pertaining to the problem solving methods (the steps they go through, when they are applicable), and how to control the whole operation to find what is faulty with a device. The types of knowledge listed in parentheses could all be contained in the knowledge section. However, to reduce the interrelations to be designed between the structural components, only three types were chosen to belong to the knowledge section: specific device domain knowledge, incremental diagnosis solution knowledge, and method applicability knowledge. A discussion of these three types is in the following paragraphs. The last knowledge type, control knowledge, is contained in the control section.

The knowledge section is divided into three main areas. This division is based on the hierarchy concept from the blackboard model and meta-level inference's concept of separation of domain and control knowledge. The first area relates to specific domain knowledge available for the diagnosis session. This could include models available for the device under diagnosis, existing empirical fault/symptom associations, component failure rates, the granularity of an acceptable solution, or computational requirements to name a few. The second area contains knowledge about the incremental diagnosis solutions. The second area is subdivided according to the three main tasks to be realized while doing diagnosis (symptom detection, hypothesis generation, and hypothesis

discrimination); incremental solution knowledge is distributed among them. Since methods are designed to satisfy the goals of tasks, dividing the second area along task lines provides specific partitions that methods (interested in satisfying the goals of only certain tasks) need to use. The design of the second area is influenced by the *diagnosis task structure*, a major find from the literature review research. Below, the design of the second area is discussed further. The third area of the knowledge section relates to the knowledge the problem solving methods use to determine their applicability. The third area is divided the same way the method section is - each method in the method section is linked to a division in the third knowledge section area that holds its applicability evaluation knowledge. The knowledge section and method section are connected in two ways: the methods read and write to the first and second areas, and each division of the third area is linked to its associated method division in the method section. Below, the design of the third area is discussed further. Figure 4.4 shows the knowledge section and its divisions as discussed in this paragraph. Table 4.3 summarizes the knowledge section contents.

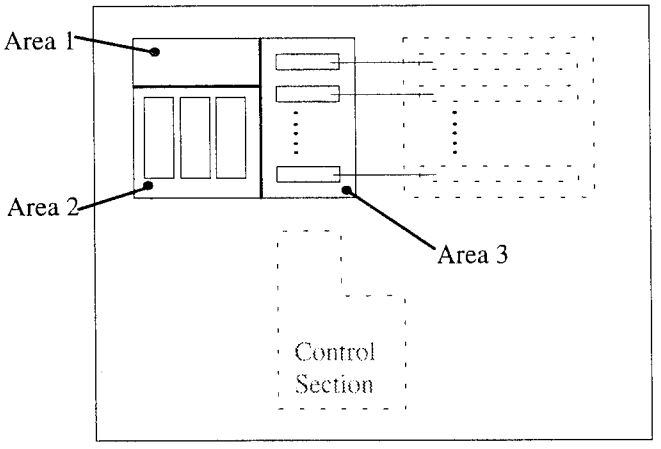


Figure 4.4 - GTS Architecture, Expanded Knowledge Section

Structural Component	Contains These Research Concepts	Filled by this Previous-System Strength		
Method Section	Domain Knowledge. Knowledge Source Independence. Knowledge Source Body.	Problem Solving Method Bodies.		
Knowledge Section	Domain Knowledge. Blackboard Data Structure. Blackboard Hierarchy Divisions. Knowledge Source Applicability Evaluation.	Area 1 Specific Domain Knowledge.	Area 2 Incremental Diagnosis Solutions. Three Subdivisions According to Task Structure	Area 3 Method Applicability Knowledge.
Control Section	Control Knowledge. Blackboard Control Module.			

Table 4.3 - Knowledge Section Contents

The second area of the knowledge section was defined to hold knowledge about the incremental diagnosis solutions. The knowledge needed to be organized to make effective use of it. The organization used for this objective was the task structure of the diagnosis process (1:45). The task structure is a graph-based representation of the knowledge surrounding the diagnosis process. The word *process* implies a series of tasks that bring about a result. Consequently, one purpose the task structure serves is to identify the diagnosis process tasks. The task structure for the multimethod approach to diagnosis contains more than just the tasks to be accomplished. Consequently, a second purpose the task structure serves is to specify the methods that can be used to complete the tasks. In summary, the task structure organizes in a graph the diagnosis process according to the hierarchy of tasks that need to be accomplished and the problem solving methods associated with each task. It is important to the knowledge section's Area two because it provides an organization for the incremental solution knowledge stored there.

An example of part of the task structure used in the GTS is shown in Figure 4.5. The complete task structure graph is contained in Appendix B.

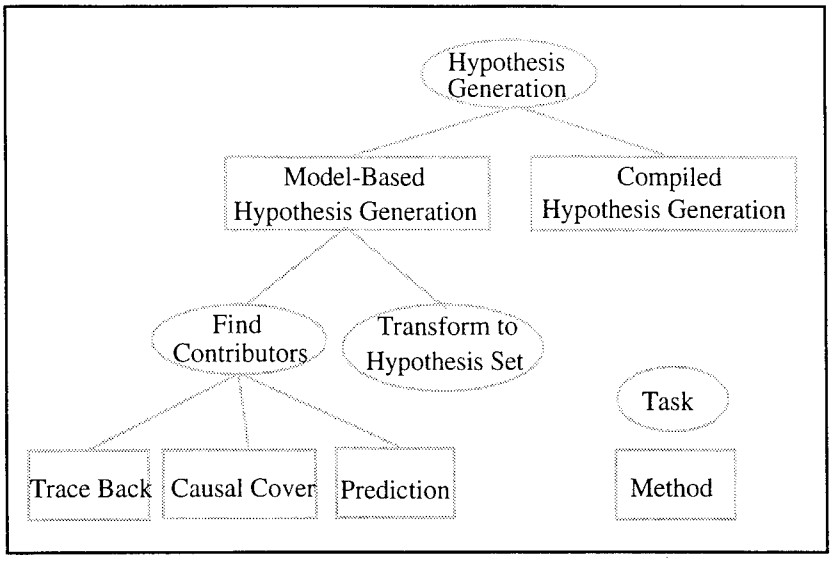


Figure 4.5 - Portion of Diagnosis Task Structure

The task structure's three highest tasks - symptom detection, hypothesis generation, and hypothesis discrimination - serve as the basis for the subdivisions in Area two. Each subdivision holds the incremental solution knowledge associated with the tasks and methods organized under the three highest tasks. Both the blackboard model and the meta-level inference model make the point that explicit representation of the knowledge used in a knowledge-based system has many advantages. This principle was used in the further design of Area two to define the kinds of knowledge to be stored in each subdivision. Once again the diagnosis task structure was used, this time to take advantage of its ability to organize the knowledge requirements of the different tasks and methods. Each method was analyzed using a modified form of IDEF-0 modeling, an industrial engineering approach to modeling processes. An IDEF-0 model decomposes a process into the functions and subfunctions that realize it. Similarly, the task structure decomposes the diagnosis process into its tasks, subtasks, and methods. For each

function, the IDEF-0 model identifies the inputs, outputs, controls and mechanisms which make the function operate. Applying the IDEF-0 model to the task structure allows the input, output and knowledge requirements to be gathered for each method and task. Applying the model consecutively up the task structure collects the knowledge requirements of lower levels into the upper levels. The knowledge requirements of the three main diagnosis tasks are then the collection of the knowledge requirements of the tasks and methods organized below them. A complete summary of IDEF-0 analysis for the diagnosis task structure is in Appendix C. Figure 4.6 shows the IDEF-0 model applied to a generic function and the modified model applied to the Find Contributors task shown in Figure 4.5. In summary, applying the modified IDEF-0 modeling technique to the diagnosis task structure allows for the explicit identification of what knowledge to store in the three subdivisions of Area two in the GTS knowledge section.

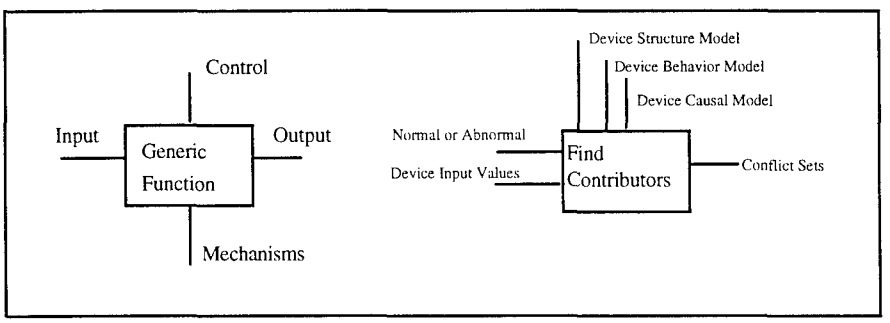


Figure 4.6 - IDEF-0 Modeling

Besides the incremental knowledge and domain knowledge, the knowledge section of the GTS architecture also deals with the knowledge each method has to describe its applicability during the diagnosis process. This is contained in Area three of the knowledge section. For each method in the method section, there is a corresponding division in Area three that is the applicability evaluation part of the knowledge source. These evaluation parts contain a *sponsor* (19:69) for each method. The sponsor lists a

method's *suitability criteria* (2:155). The suitability criteria provide a means to explicitly capture the requirements the methods have to operate.

According to Benjamins, the suitability criteria can be categorized along three axes: the type of knowledge axis, the strictness axis, and the persistency axis (see Figure 4.7) (2:155-157). The type of knowledge axis is further divided into four areas: epistemological, environmental, computational, and assumption areas. The epistemological area specifies the type of domain knowledge the method needs to have available. Examples include whether the device model is based on structure and behavior or causal associations, if the model is static or continuous, or what is the scope of the model (15:17-20). The environmental area specifies physical conditions that need to be present with the device under diagnoses for the method to work. An example is a method that probes the device to gather observations requires the probe points be accessible. The computational area reflects the characteristics that relate to a method's processing requirements- memory requirements, time requirements, etc. Last, the assumption area captures the implicit assumptions built into the methods of past diagnostic systems. For example, some methods only produce a hypothesis containing a single component as an explanation for a fault. If multiple component combinations are desired for a given diagnosis session, a method adopting the single fault assumption cannot be used. The strictness axis is divided into two areas: necessary and useful. A criterion categorized as necessary absolutely needs to be present for the method to operate. A useful criterion specifies conditions that would aid the method's operation, but would not prohibit the method's operation if it were not available. In evaluating its applicability, a method must first satisfy all its necessary criteria before even looking at the useful ones, for if the necessary are not satisfied, the method is guaranteed to not operate. The useful criteria are helpful in discriminating between several applicable methods. The third axis, the persistence axis, is divided into two areas: static and dynamic. Static criteria are those

that describe knowledge that does not change throughout the diagnosis process, whereas dynamic criteria knowledge can change.

Each criterion a method has can be evaluated to see what criteria knowledge areas apply to it. For example, for the above example regarding probe point accessibility, the Environmental, Necessary, and Static criteria areas apply. Evaluating all method criteria to find what areas apply allows for a consistent, explicit representation of suitability knowledge, and allows for comparisons between methods to make dynamic method selection choices.

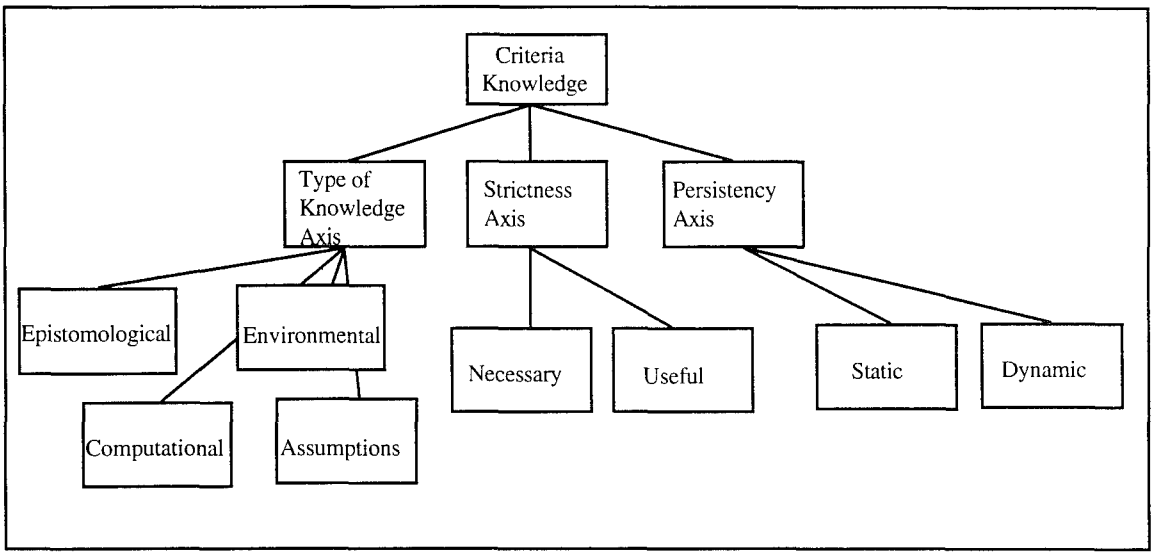


Figure 4.7 - Suitability Criteria Knowledge Axes and Areas

This concludes the design description for the knowledge section of the GTS architecture. Summarizing, the knowledge section's job is to provide an organization for the diverse types of knowledge used in the GTS. It holds knowledge that deals with the incremental diagnosis solutions, specific domain knowledge, and method applicability. It was designed considering the importance of separating domain from control knowledge, and augmented the basic blackboard problem solving model with several features from

past multimethod approaches to automated diagnosis. With the methods and knowledge sections described, the control section will now be presented.

4.2.2.4 Control Section. In terms of a meta-level inference system, the method and knowledge sections are at the *object level.* They deal with specific domains and are tightly focused. The control section, on the other hand, is at the *meta-level.* The control section for the GTS has knowledge of how to use the object level knowledge. The control section is where decisions are made that guide the diagnosis process.

The control section uses the same strategy of using components from other automated diagnosis systems just as the method and knowledge sections did. There are three concepts that go into the design of the control section (see Figure 4.9 and Table 4.4). The first is the task structure already mentioned in the knowledge section. There, it was used because it organized the knowledge used during the diagnosis process. In the control section, the task structure is used for how it organizes the diagnosis tasks. The control section uses the task structure to keep track of where the system is in the diagnosis process. Also, the task structure provides the control section with the methods and tasks directly applicable to the state of the process. It focuses the control section regarding what to consider doing next; it does not need to consider all methods and all suitability criteria but only those directly related to the current diagnosis task. The second component of the control section is the *selector* (19:69). The selector receives the set of methods and tasks directly applicable to the state of the process and polls the sponsors contained in the knowledge section Area three. Upon receiving the applicability knowledge from the methods, the selector calculates the applicability scores, selects the highest scoring method, and invokes the method to start operating. The third component, as currently designed for the GTS, is a simple loop that repeatedly chooses and applies the most appropriate problem solving method until a suitable diagnosis is found The pseudo code for this loop is in Figure 4.8. Figure 4.9 shows the expanded control section, and Table 4.4 summarizes the contents of the Control section.

```
While suitable diagnosis not found
        Based on the current diagnosis task
                Find the methods associated with the task
        While diagnosis tasks goals not met
                If no associated methods then
                        Announce Diagnosis Failure
                        End
                End if
                Evaluate method applicability
                Select the most appropriate method
                Invoke the method
                Evaluate method results
                        If results do not satisfy diagnosis task goals then
                                Remove current method from consideration
                        End If
        Loop
        If suitable diagnosis not found then
                Advance to next diagnosis task
        End if
Loop
Report suitable diagnosis
```

Figure 4.8 - Control Loop Pseudo Code

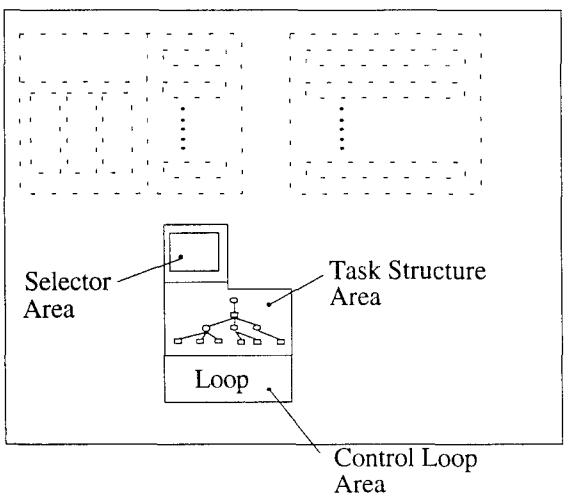


Figure 4.9 - GTS Architecture, Expanded Control Section

Method Section	Domain Knowledge. Knowledge Source Independence. Knowledge Source Body.	Actual Problem Solving Method Bodies.		
Knowledge Section	Domain Knowledge. Blackboard Data Structure. Blackboard Hierarchy Divisions. Knowledge Source Applicability Evaluation.	Area 1 Specific Domain Knowledge.	Area 2 Incremental Diagnosis Solutions. Three Subdivisions According to Task Structure.	Area 3 Method Applicability Knowledge (Sponsors) Suitability Criteria.
Control Section	Control Knowledge. Blackboard Control Module.	Task Structure Area Task Structure	Selector Area Selector	Loop Area Control Loop

Table 4.4 - Control Section Contents

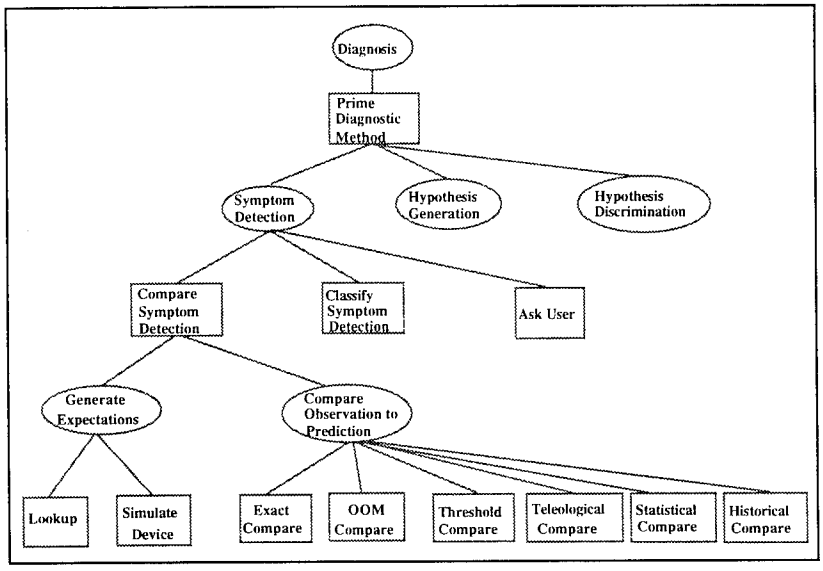


Figure 4.10 - Symptom Detection Portion of Task Structure

4.2.3 GTS Operation Example. The previous sections described in detail the design architecture for the Generalized Troubleshooter. While those sections described the structure, this section briefly describes the operation - how a diagnosis could be formulated using the architecture. This example simulates a consistency-based approach

for only the processing done for the Symptom Detection task. This is sufficient to show how all the components of the GTS architecture operate and to keep the discussion to a minimum. Figure 4.10 shows the Symptom Detection portion of the Diagnosis Task Structure. This figure aids in visualizing the operation described below.

For this diagnosis session, several assumptions are made: structural and behavioral models are available, there is at least one discrepancy between model predictions and the observations, and the knowledge required by the Exact Compare method is available. The last assumption is applicable when performing the "Compare Observation to Prediction" task. To begin, Area one of the knowledge section would contain the device specific structural and behavioral models. Assuming the troubleshooter is continuously on, the control section starts off in the Diagnosis task. Using the task structure, it would find that the Prime Diagnostic Method is the only method associated with the Diagnosis task. The Control Loop then tasks the Selector to poll the Sponsor for the Prime Diagnostic Method. The Sponsor reports to the Selector its applicability knowledge. The Selector evaluates the applicability knowledge using the knowledge in the knowledge section to compute an applicability score. The applicability score shows the method's criteria are satisfied. The Selector then selects and invokes the Prime Diagnostic Method. At this high level in the task structure, the Prime Diagnostic Method does no real work. It merely confirms that the diagnosis process should continue. Upon evaluating the method results, the Control Loop evaluates the method as successful, decides that no diagnoses have been found and consequently updates the process task position to the Symptom Detection task. This is the end of one pass through the control loop.

Using the task structure, the Control Loop finds that applicable methods are the Compare method, the Classify method, and the Ask User method. This means to satisfy the goals of the Symptom Detection task, the observations and predictions need to be compared and then classified as normal or abnormal, or the GTS user needs to tell GTS which observations are abnormal, bypassing the Compare and Classify methods. Although

not explicitly specified in the control section at present, because the task structure is not an and/or graph, this and/or sequence knowledge must be taken into account. Chapter five discusses how this may be possible. The Selector polls the Sponsors for these three methods and they report back their applicability knowledge to the Selector. The Selector determines that the knowledge needed by the Compare and Classify Methods is present in the knowledge section. The Selector invokes both methods with the Compare method scheduled first. Again since the Compare method is at a high level in the task structure, no real work is done although the method is evaluated as successful because it can be applied to the current situation. The Control Loop decides no diagnoses have been found and updates the process task position to the tasks organized under the Compare method - the Generate Expectation and Compare tasks. The Classify method, not having completed yet, is kept on hold, ready to run when all task goals preceding it in the task structure are completed. This is the end of the second pass through the control loop.

Since the current process task position is the Generate Expectation task, the Lookup and Simulate methods are identified as being the ones to consider using next. The Selector polls the Sponsors for these two methods and here is when some choices are made. Considering the domain knowledge in Area one, the Selector evaluates Lookup to "not applicable" since no domain-specific database is available that tells what the device's output values should be based on the applied inputs. As a result, the Simulate method evaluates to "applicable" and the Selector invokes the Simulate method. The method reads from Area one the device inputs, and writes the values predicted for the internal points to the Symptom Detection subdivision of Area two. Assuming no problems are encountered, the Control Loop evaluates the methods as successful, decides no diagnoses have been produced yet, and updates the current process position to the Compare task. The Classify method is still on hold.

The task structure identifies a number of methods possibly applicable to the Compare task and has the Selector poll the Sponsors for the Exact, Order-of-Magnitude,

Threshold, Teleological, Statistical, and Historical methods. Because of the assumption that the knowledge required by the Exact Compare method is available, the Selector evaluates the Exact method to "applicable". Each of the other methods under consideration requires specific knowledge that is not available. The Selector invokes the Exact method, and it proceeds to compare the predicted values to the observed values, noting the values that are not exactly equal. It places its results in its portion of the Symptom Detection subdivision of Area two.

The Classify method that has been on hold for the last few cycles is now invoked. The job of the Classify method is to evaluate the comparisons performed by the Compare method and assign a value to them of either normal or abnormal according to the method used to compare predictions to observations. Since the Exact Compare method was used, any prediction/observation comparisons that was not exactly equal is classified as abnormal. Assuming there is at least one discrepancy, the Classify method writes the abnormal classification to its part of the Symptom Detection subdivision of Area two. Still no suitable diagnosis has been found, so the current process position is updated to the Hypothesis Generation task and the diagnosis process continues.

This example of the diagnosis processing through one branch of the task structure demonstrates the general operation of the GTS. For each task, a breadth first search determines the methods available to choose from. In the times when more than one method is chosen to be accomplished, a depth first search is performed from the first method scheduled. Using the task structure in this way guides the Selector's search for methods to apply in that not all the methods need to be evaluated at every step during the process. The diagnosis process ends when there are no more methods to invoke and the results of the string of methods invoked in the past have all been successful. At this point, the suitable diagnosis will be in its portion of the Hypothesis Discrimination subdivision of Area two and the current process position is returned to the diagnosis task. The diagnosis process also ends if all the methods applicable to a task are executed unsuccessfully. This

means a diagnosis task cannot be completed with the set of methods available for the particular diagnosis problem. Since a task failed completion and there are no others to try the overall diagnosis process fails. If the full troubleshooter was included and the diagnosis process successfully completed, it would take the generated hypotheses and formulate a corrective action plan, but for now this is outside the scope of the GTS.

4.3 Design Conclusion

This chapter has presented the product of the research performed for this thesis: an architecture for a Generalized Troubleshooting system based on the dynamically selected multimethod approach to diagnosis. Its main features include a blackboard problem solving model augmented with specific concepts gathered from other multimethod diagnosis systems. The architecture consists of three structural components: the method section, the knowledge section, and the control section. The method section holds the steps to be performed by various methods that each work on a narrow range of problems. The knowledge section holds the domain specific knowledge required by the methods to operate. The control section holds the knowledge needed to make decisions about when to apply the method knowledge. Because of the separation of domain and control knowledge, the GTS has meta-level inference characteristics, meaning it uses knowledge about domain knowledge to guide the selection of problem solving methods during the diagnosis process.

V. Analysis

5.1 Chapter Overview

This chapter examines the GTS system and comments on its strengths and weaknesses. Possible improvements to the architecture will be presented, and the role of the GTS in the targeted application - the MAGIC program - is discussed.

5.2 GTS System Strengths

Because the GTS architecture is based on the blackboard model, it inherits many of the advantages the blackboard has. One advantage is the modularity the blackboard model provides. A well planned modular design provides a well organized framework in which to put a system's elements connected through specific interfaces. Also, it is generally more acceptable to focus on a narrow range of a problem and put different pieces together for the overall solution than to try to make one module fulfill all requirements on its own. Modularity is used often in the GTS system, starting at a high level with the architecture sections and continuing with the subdivisions within them. In the knowledge section, modularity allows the specific identification of the type of knowledge being used. This is important when gaps in the system appear. Modularity can help in tracing the missing link to a narrow range of suspect modules and once the suspect is located, the things to consider to fill the gap in knowledge is reduced to investigating a similarly narrow range. Modularity also assists in separating the types of knowledge, an important consideration when using meta-level inference systems. Other beneficial features of a modular design include easier testing since testing a module with a manageable set of requirements for correct operation is easier than testing an entire

conglomeration of system level requirements. Maintenance is generally easier also because the internals of the modules may be modified with smaller chance of introducing errors into other parts of the system. Another blackboard attribute that the GTS inherits is the ability to assemble and dynamically control various specialized problem solvers. This attribute is exactly what the GTS approach encompassed, so there is a good match between the blackboard's capabilities and the requirements the GTS needs to fulfill.

A second GTS system strength is the architecture's ability to store and use widely varying types of knowledge to solve a diagnosis type of problem. The architecture promotes experimentation to compare, for example, different sets of problem solving methods, different method selection control strategies, or the effects of different types of knowledge on the diagnosis process. The architecture modules provide "parameters" which can be altered in different GTS configurations to determine the merits of each configuration.

A third system strength is the GTS' robustness. It is robust in terms of the different devices it can be applied to, the range of situations where the deep knowledge it uses is applicable, and its ability to recover from an unproductive line of reasoning. Regarding devices, the same diagnosis approach can be applied to different devices by loading the GTS with domain knowledge (mainly models) about them. As long as different devices can be described in modeling terms consistent with the knowledge formats the problem solving methods use, the same generic method can reason about them. The deep knowledge robustness means that deep knowledge (electrical component operation equations, for example) that applies in one situation also applies in another where the same device is used. A common knowledge base can be called upon for many devices, precluding the need to derive all the domain knowledge surrounding a device every time a new one is created. Last, unlike an algorithm that may quit if the problem data does not exactly match the data the algorithm expected, the GTS allows for attempts at finding a solution to continue even though one method did not successfully complete.

5.3 GTS Shortcomings and Improvements

Although the GTS seems to have significant advantages over past automated diagnosis systems, it is not the final word on multimethod diagnosis. The following paragraphs discuss some shortcomings I see with the GTS architecture and suggest ways to overcome them. Expanding the architecture also helps to overcome the limitations made by the focusing assumptions discussed in Chapter three, section two.

The task structure intuitively appears to be incomplete because it is based on an analysis of the actions performed by human troubleshooters. The high level tasks of what a troubleshooter tries to accomplish during diagnosis are relatively constant, but the means he uses to realize the task goals will likely change. Therefore, the task structure would require periodic updating to keep pace with the state of the art. For example, in my literature search, I found at least one method not currently captured in the task structure. It is called the *iterative search method* (15:22). In this method, a model of a device is used to predict component values and a discrepancy detector finds the differences between predictions and observations. The method then searches the space of model variations it has as domain knowledge to find the model of incorrect behavior whose evaluation predicts the observations. The hypotheses for fault causes are then the differences between the model of all correct behavior and the model of incorrect behavior. Based on the literature, the model-based diagnosis field appears to have an abundance of ongoing research, and the current task structure should reflect the results of new research.

Not only should the task structure be updated with new methods, but it should have additional reasoning paradigms added too. One idea behind multimethod diagnosis was to capitalize on the combined use of individual methods allowing methods to cover the shortcomings of each other. I feel this notion of combined use should be extended to reasoning paradigms also. Where one reasoning method falls short, perhaps there is another that can take its place. An excellent example of a multi-paradigmed system is

SRS discussed in Chapter two. Architecturally, additional reasoning paradigms could be introduced as additional knowledge and method sections still under control of the control section. As the knowledge section is subdivided into a hierarchy of knowledge, the knowledge and method sections associated with another reasoning method form a hierarchy of blackboards in the new GTS system. Additionally, considering the improvement in the previous paragraph, the methods of the new reasoning paradigms would be analyzed to see what tasks of the existing task structure they support. Conceptually, inserting those methods into the task structure would require simply inserting the methods under their appropriate tasks. Architecturally, this translates to filling in the new knowledge and method sections for the new paradigms just as was done for the model-based paradigm. Finally, it may be found that other reasoning paradigms may not only add methods to choose from in accomplishing a task, but may identify additional tasks themselves. Reasoning paradigms other than model-based reasoning appear to be another source of knowledge to add to the pool already being used by GTS.

A generalized troubleshooter aims to be efficient as well as robust in developing diagnoses. One shortcoming of the current GTS architecture is that the control section does not have insight into the problem solving methods *as they run* to evaluate if the method is converging on expected results. Currently, the problem solving method reports to the control section after it has completed and its success is determined then. If the method failed, and started failing an appreciable amount of time before its planned end, then the time spent diverging from expected results serves no useful work. Perhaps a way to evaluate if a method is diverging from a solution is to have GTS keep a database of performance metrics based on past cases where the method was used. The past cases may provide a reference the control section can use to make decisions about a method's current progress. Also, perhaps GTS could be recursively used on itself. Just as a device is modeled and diagnosed when observations do not match predictions, perhaps the problem solving methods themselves could be modeled and evaluated by GTS (while they are

running), leading to possibly diagnosing problems with a method when its performance does not match expectations. This would also expand the entities GTS deals with from just physical entities to abstract entities since a method says *how* to do something. To control a method while it is running, there need to be communication between the control section and the problem solving method so that method success can be evaluated during method execution and control commands can be issued if needed.

Currently, GTS has little latitude in the order of the diagnosis tasks it performs. As discussed in Chapter four, the current diagnosis process state follows a depth first search through the tasks in the task structure. Knowledge of the sequence of diagnosis tasks is partially defined by the layout of the task structure, similar to how rule selection is sometimes implicitly defined by the ordering of the rules in a production system rule base. As seen in the operation example, some methods depend on other methods being performed first, and this sequencing knowledge in not currently available to the control section using the task structure alone. To overcome this deficiency, knowledge regarding task sequencing needs to be explicitly specified. This could possibly be done by adding meta rules to the control section, or possibly by adding the sequencing knowledge to a method's suitability criteria. Specifying the sequencing knowledge is consistent with GTS' design approach to explicitly represent control and domain knowledge, but doing this still leaves GTS with a rigid sequence of tasks the diagnosis process must go through each time, and perhaps the sequence does not need to be the same for each diagnosis problem. Task sequencing seems like an area that can be generalized more so that knowledge of where to go in the diagnosis process may be used like all the other knowledge in the GTS system to reason about what is the most appropriate task to do next.

Last, the current GTS architecture is targeted to diagnose electronic devices on board a satellite. These are physical entities. However, diagnosis can be performed on non-physical entities too, like the problem solving methods themselves previously mentioned. An interesting extension to GTS would be to diagnose something abstract

such as a process in an office or in manufacturing. A GTS acting in this capacity could make an excellent program management tool to monitor and control program execution, increase process efficiency, or aid in program planning by allowing processes to be simulated and debugged before implementation. Using the GTS on an abstract entity would give some measure of its domain independence. Going the other direction, project management concepts could aid the GTS. Specifically, project management principles (involving decision-making, for example) could be incorporated as heuristics in the GTS control section as another source of knowledge for how to guide method selection.

5.4 GTS Target Application

The work done in developing an architecture for the GTS was not just a design exercise. It was developed with a particular application in mind. Chapter three discussed the ongoing MAGIC satellite control program whose requirements influenced many of the choices made for the GTS. The MAGIC program has these goals: provide an intelligent system to allow satellite operators to manage multiple satellite constellations and block releases, be easily expanded for new constellations and block releases, be easily modifiable, and reuse domain models. The first step of MAGIC is TAS, or Telemetry Analysis System. TAS processes raw satellite telemetry to check for out of limit conditions for satellite parameters, provides telemetry plots and graphs, and offers trending analysis. No diagnostic capabilities are planned for TAS. MAGIC step two expands on step one to make a decision support system. This system will analyze out of limit conditions and other system information to determine if a known anomaly exists, what its predefined solution plan is, and present to the operator the solution for operator approval. Step three goes even further to allow the expert system to determine causes of known and unknown anomalies, and aid the operator in how to resolve the problem. Step three is about troubleshooting. It is the last step that I feel the GTS can fulfill the best.

Since diagnosis is just half of troubleshooting, to fully satisfy the goals of troubleshooting, the GTS could be augmented with a planning system. The planning system would take the diagnoses from GTS as input and produce for output a plan for the sequence of steps to perform to clear the trouble. Once causes for faults on board a satellite are found, commands to correct the faults need to transmitted back to the satellite. A troubleshooting expert system used this way could conceivably diagnose problems and execute corrective action plans without the need for satellite operator involvement. In light of current policy to reduce the training and rank of satellite operators, and the reduction or elimination of expert troubleshooting teams for satellite anomalies, the GTS and its extensions offer an approach to capture the expertise once applied by human operators and engineers. GTS can augment a gap created by these policy changes to keep valuable satellite resources at operational levels.

5.5 Analysis Conclusion

This chapter has looked at the GTS architecture to comment on its strengths, weaknesses, possible areas for improvement, and how it may contribute in its foreseen target application of an intelligent control system for satellite operation. I feel the overall conclusion is the GTS architecture provides a sound basis to reuse existing artificial intelligence technology and diagnosis research results on which to implement the diagnosis portion of an automated troubleshooting tool.

VI. Conclusion

This thesis dealt with *troubleshooting*, the process of 1) determining causes for why an entity is not behaving correctly and 2) formulating action plans to alleviate those causes in order to return the entity to correct operation. The process of determining causes for faults is called *diagnosis* and was the portion of the overall troubleshooting process that this thesis focused on.

Diagnosis transforms initial observations citing incorrect behavior into reasons explaining the behavior. The transformation can be viewed as a *process* in which a series of steps are applied to incrementally develop an explanation solution. One way to organize a process is to identify its constituent components, the goals they aim to achieve, the actions performed in them, the resources they require, and their relative ordering. This thesis adopted this view of the diagnosis process and therefore set out to define an architecture for automated knowledge-based diagnosis in terms of the tasks the process performs, the goals the tasks have, the methods used to satisfy the goals, the knowledge required to perform the methods, and the control of method selection. Furthermore, this thesis assumed that 1) there is in general more than one method to use to achieve the goals of a task and 2) that the choice of which method to use may be flexible as long as the task goals are completed. The overall system is called the Generalized Troubleshooting System (GTS) and the architecture developed for this thesis supports the first subgoal of troubleshooting, efficiently finding explanations for faulty device behavior.

The approach to defining this architecture involved combining the individual strengths of existing diagnosis technologies into a common framework. The architecture is based on the blackboard problem solving model, an approach to problem solving that opportunistically combines the expertise of various domain experts to solve a larger

problem that no expert could solve alone. The blackboard provides the conceptual model which to implement the individual strengths of other diagnosis systems.

This research produced a number of results. Of prime importance is the task structure of the diagnosis process, a graph organization of the incremental tasks to be performed and sets of methods useful to realize the tasks. The task structure provides a foundation to organize the knowledge required by the methods and the control of the diagnosis process. Another important result is the identification of the different types of knowledge the methods in the task structure use. Some of this knowledge dictates when the problem solving methods are applicable, and a third important result is the organization of this type of knowledge into a common format for all methods. This provides a means to control the diagnosis process by reasoning about when to invoke a method. The last important result is the apparently successful merger of the task structure with the blackboard problem solving model. The tasks to be performed during the dynamic method selection approach to diagnosis line up well with the functionality the blackboard model pieces provide.

The results of this research could be expanded in a number of ways. The first involves making an implementation of the architecture. No implementation was made for this work; however, the architecture provides a design upon which a future implementation could possibly be based. Implementing a system based on the GTS architecture and comparing its performance with past automated diagnosis systems would give validity to the design choices. As with most first-time designs, I feel there is room for improvements in the GTS architecture itself. The areas I feel should be expanded include continuing the idea of combining domain specific experts by introducing other reasoning paradigms and their methods into the task structure to augment the capabilities of the model based methods. The next area I feel should be explored is enabling the dynamic selection of methods to be made anytime when a problem solving method is operating

instead of waiting for method completion. The addition of these two areas would increase the tools available to the GTS to determine a diagnosis, and attempt to curb an increase in processing time the extra reasoning methods may introduce.

APPENDIX A. GTS Task and Method Descriptions

This appendix contains a brief description of the functionality for the methods and tasks used in GTS. The method/task numbers in tables A-1 and A-2 refer to the identification number of the nodes in the task structure graph contained in Appendix B.

Method Number	Method Name	Brief Description
R1	Prime Diagnostic	Encapsulates 3 main diagnosis subtasks.
R2	Compare	Generates expected value and compares to observation.
R3	Classify	Classifies comparison as normal or abnormal.
R4	User	User decides normal or abnormal.
R5	Model Based Hypothesis Generation	Generate causes by comparing model predictions to observations.
R6	Compiled	Finds causes by using compiled symptom/cause associations.
R7	Discrimination	Narrows list of possible hypotheses.
R8	Lookup	Uses database to determine expected values.
R9	Simulate	Compute expected values by using simulation model.
R10	Exact	Compares exact values of predictions and observations.
R11	Order Of Magnitude	Compares OOM's of predictions and observations.
R12	Threshold	Checks for difference of predictions and observation to exceed a threshold value.
R13	Teleological	Compares according to function or purpose.
R14	Statistical	Expected values expressed in distribution terms.
R15	Historical	Uses historical data to decide if an observation is normal or abnormal.
R16	Trace Back	Uses structural model to find components linked to a discrepancy between predicted and observed value.
R17	Causal Covering	Uses causal model to cover observations with symptom causes.
R18	Prediction	Stores names of components contributing to a constraint network point calculation.
R19	Set Cover	Uses candidate component sets to cover symptom set.

Table A-1 - GTS Method Descriptions

R20	Intersection	Intersect all candidate component sets to find candidates that could cause all symptoms
R21	Subset Minimality	Build candidate sets that do not have another candidate set as a proper subset
R22	Cardinality Minimality	Build sets of candidate sets implicating fewest number of failing components
R23	Constraint Suspension	Remove model of suspected component from device model while testing hypotheses
R24	Corroboration	Removes a fault hypothesis if it predicts values for its components that differ from observations
R25	Fault Simulation	Instantiate a device fault model. Simulate to match predictions with observations
R26	Random	Select at random a hypothesis from the set of hypotheses to collect further data for
R27	Smart	Select a hypothesis from the set of hypothesis by associating each with a measurement cost
R28	Compiled Test	Consult database of compiled test procedures to collect further data for hypotheses
R29	Probe	Probe the device at test points to collect further data for hypotheses
R30	Manipulate	Determine a new input vector to apply. See if implied faulty components same under new inputs
R31	Replace	Replace suspected components and apply old input vector . Successful if symptom disappears.
R32	Interpret In Isolation	Keep/reject hypotheses based on additional collected data
R33	Split Interpret	Split hypothesis set into two parts. Reject all hypotheses that do not predict newly collected data
R34	Based On Local Costs	Select hypothesis to test based on probability of failure data for a suspected component
R35	On The Number Of Tests	Select hypothesis to test based on expected number of tests to isolate cause
R36	Based On Overall Costs	Select hypothesis to test based on a combination of local costs and number of tests.
R37	Direct Measure	Collect additional discriminatory data directly form test points.
R38	Indicator Based	Use the value at a directly accessible test point to infer the value of another non accessible test point

Table A-1 - GTS Method Descriptions (Continued)

Task Number	Task Name	Brief Description
C1	Diagnosis	Determines possible causes for abnormal device observations.
C2	Symptom Detection	Determines if observations compared to predictions are abnormal.
C3	Hypothesis Generation	Generates possible causes for explaining normal and abnormal observations.
C4	Hypothesis Discrimination	Discriminate among possible causes using additional observations.
C5	Generate Expectation	Generate expected values for given inputs.
C6	Compare	Compares expected values to observed values
C7	Classify	Classifies observations as abnormal or normal.
C8	Ask User	Prompt user to classify observation as abnormal or normal.
C9	Find Contributors	Finds components that contribute to an abnormal observation.
C10	Transform To Hypothesis Set	Transforms contributor sets to hypothesis sets.
C11	Prediction Based Filtering	Removes hypotheses from consideration that predict observations the device is not showing.
C12	Abstract	Refers to compiled symptom/cause database.
C13	Associate	Associates causes with their symptoms.
C14	Probability Filter	Predicts observations that should be observed if the causes exist.
C15	Select Hypothesis	Select a hypothesis that requires additional observation data.
C16	Collect Data	Collection additional observation data.
C17	Interpret Data	Updates the hypothesis set based on the additional data.
C18	Lookup	In table, look up output values for given input values.
C19	Simulate	Uses model to determine output values based on input values.
C20	Equal Check	Compares prediction to observation for equality.
C21	Determine Order of Magnitude	Generates order of magnitude for predictions and observations.
C22	Determine Ratio	Determines ration of predicted value to observed value.
C23	Check Against Threshold	Compares difference between observation and prediction to threshold knowledge.
C24	Teleological Abstract	Determine design purpose for a component.
C25	Statistical Compare	Compare observations to an expected probability distribution.
C26	Historical Compare	Compare observations to past values for the same parameter.
C27	Find Upstream	Determine components that are physically connected to a suspect component.
C28	Causal Covering	Determine what possible causes predict the observations.
C29	Set Cover	Apply set cover algorithm to determine if suspected components could explain all abnormal observations.
C30	Intersect	Intersect all suspect component sets to find commonalities.
C31	Subset Minimality Cover	Find the smallest subset of suspect components that explain the abnormal observations.
C32	Cardinality Minimality Cover	Find the smallest number of components that are suspects.
C33	Select Random	Select a random element from a list of hypotheses.
C34	Suspend Constraint	Remove a component's model from the device model. Used in hypothesis generation.
C35	Check Consistency	Check if observations are consistent with the model predictions.

Table A-2 - GTS Task Descriptions

C36	Delete	Remove a hypothesis from a list.
C37	Select Fault Model	Select a fault model for the device to simulate. Used to compare the predictions to observations. If there is a match, the model tells how the device is failing.
C38	Estimate The Cost Of Testing The Hypothesis Set	Gather cost data to determine for which hypothesis to gather additional data.
C39	Order Hypothesis Set	Order hypothesis set based on cost estimates.
C40	Select First	Select first hypothesis from list of ordered hypotheses.
C41	Compiled Test	Run a test script to collect additional observation data.
C42	Measure	Measure the value at a probe point.
C43	Deduce Input Vector	Determine new inputs to device to exercise suspect components.
C44	Replace Hypothesis	Replace a suspected component with a new component.
C45	Split Hypothesis Set	Remove half the hypothesis set from consideration based on the result of additional observations.
C46	Obtain	Obtain new observations after applying new inputs.
C47	Estimate Local Cost	Use component a prior failure probabilities to rank suspects.
C48	Estimate Number Of Tests	Use information entropy to estimate the number of additional measurements required to isolate a fault starting with a give suspect component.
C49	Estimate Overall Costs	Combine Local Cost and Number of Test estimates to form a new estimate.

Table A-2 - GTS Task Descriptions (Continued)

APPENDIX B. Diagnosis Task Structure

This appendix details the task structure of the diagnosis process. The task structure is a graph-based representation of the knowledge for the diagnosis process. One purpose the task structure serves is to identify the tasks of the diagnosis process. A second purpose is to identify the problem solving methods that can be used to complete the tasks. Both tasks and methods are represented by nodes of the graph. A task has a goal and dictates what needs to be done for a certain stage of the diagnosis process. A method is associated with a task (via an arc) when performing the method can satisfy the task's goal. Tasks are decomposed into methods, which are in turn decomposed into the tasks necessary to perform the method, to form alternating task/method levels down the task structure hierarchy. The recursive decomposition ends when primitive inference tasks are identified. Primitive inference tasks, analogous to subroutines or object methods, are specific procedures that operate on domain knowledge to perform a specific job. Finally, some methods rely on tasks that are not in the hierarchy below them. Therefore, the task structure is not strictly hierarchical. In summary, the task structure organizes the diagnosis process in a graph according to the hierarchy of tasks (goals) that need to be accomplished and the problem solving methods associated with each task.

One use for the task structure is to control the diagnosis process. Traversing the task structure according to its links steps through the tasks in the order necessary to solve a diagnosis problem. Furthermore, for each task, the applicable methods that can achieve its goals are directly linked to it. This focuses the controller's search for applicable methods. The controller does not need to search through *all* methods identified in the task structure, but only those linked to the current task. A second use for the task structure is to organize the knowledge required at each stage in the diagnosis process. This is necessary to dynamically choose the most applicable method to perform at each

stage. Each task and method has certain knowledge requirements that must be satisfied for it to operate. For each node, representing a task or method, its knowledge requirements are the sum of the knowledge requirements for the lower level nodes linked to it. In summary, the task structure can be used to control the diagnosis process. It provides a means for the generalized troubleshooting system (GTS) controller to step through the diagnosis tasks in the order required to solve a diagnosis problem. Furthermore, it provides a means to allow the knowledge requirements to be explicitly identified for each stage of the process, contributing to dynamic method selection.

The following pages show the diagnosis task structure used in the GTS. Its purpose here is to see in one picture the entire task structure. Tables B.1 and B.2 list the names for the tasks and methods, respectively. Table B.3 lists the links between tasks and methods. The "Node ID number" for each name corresponds to the number of the appropriate node on the graph. Tasks are represented by circle nodes, and methods are represented by rectangular nodes. The graph itself is divided over two pages. Page B-3 contains the left half of the graph, and page B-4 contains the right half. The following picture demonstrates how to join the two parts.

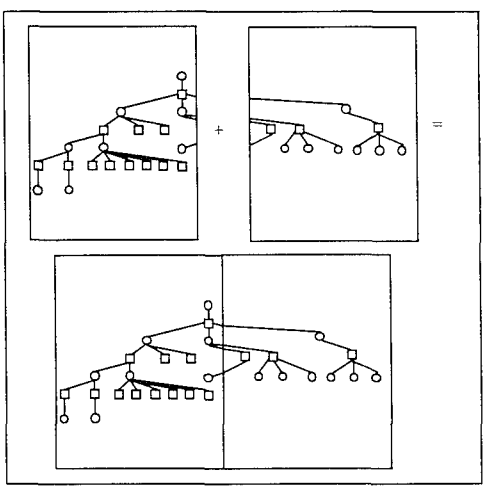
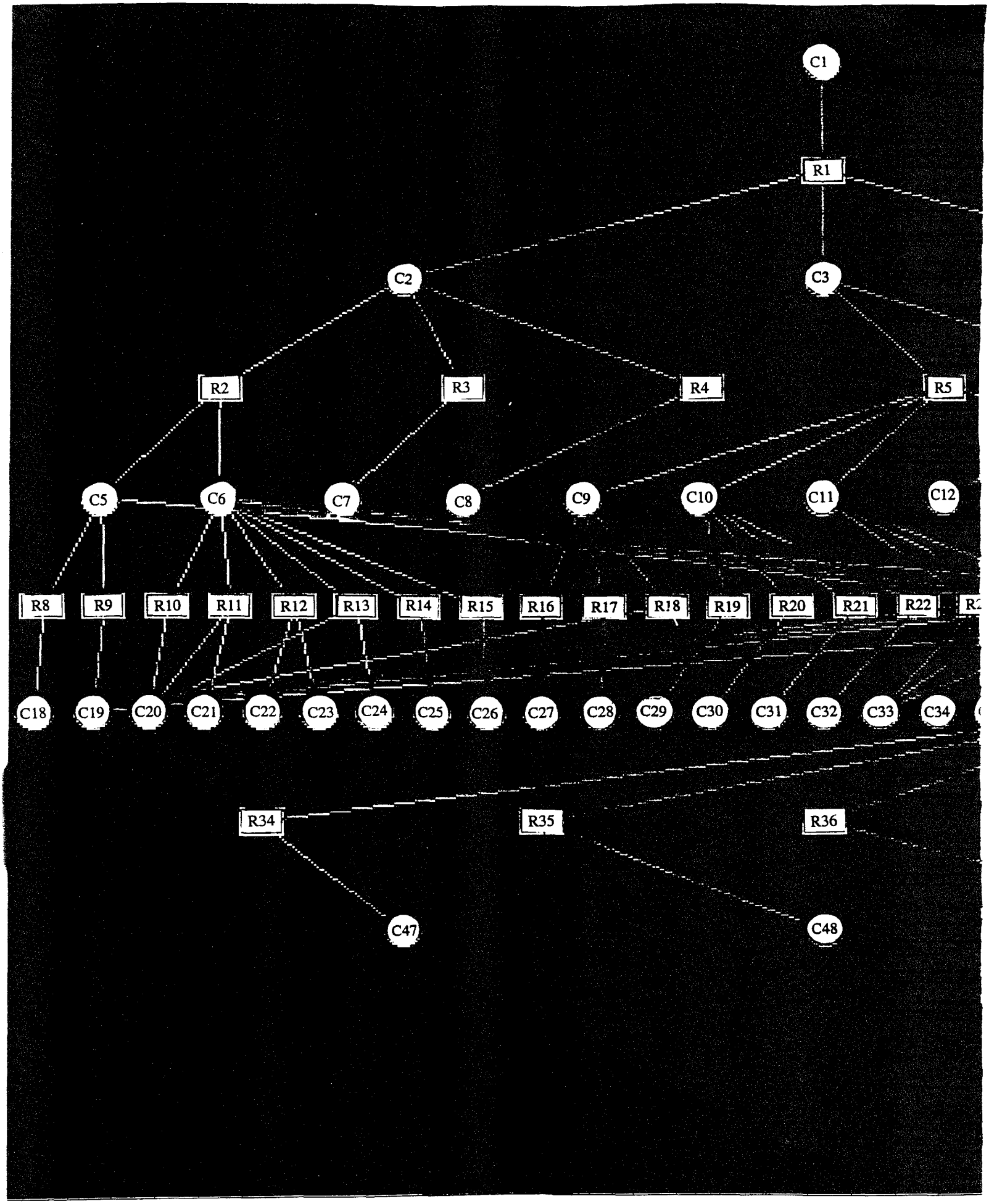
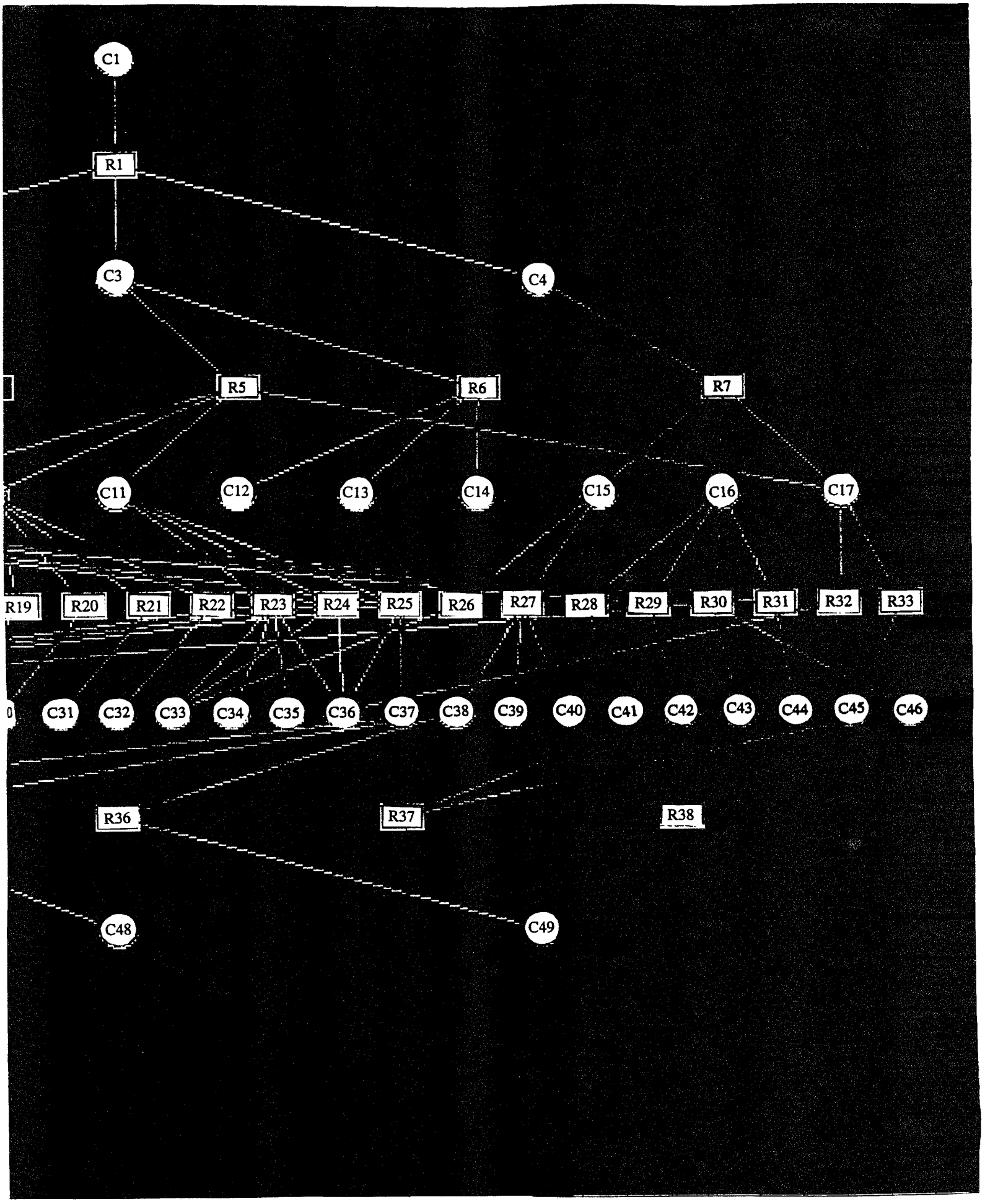


Figure B.1 - Task Structure Construction


C1
R1
C2
C3
R2
R3
R4
R5
C5
C6
C7
C8
C9
C10
C11
C12
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
C18
C19
C20
C21
C22
C23
C24
C25
C26
C27
C28
C29
C30
C31
C32
C33
C34
R34
R35
R36
C47
C48


C1
R1
C3
C4
R5
R6
R7
C11
C12
C13
C14
C15
C16
C17
R19
R20
R21
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31
R32
R33
C31
C32
C33
C34
C35
C36
C37
C38
C39
C40
C41
C42
C43
C44
C45
C46
R36
R37
R38
C48
C49

Node ID number, Circle Nodes	Node Name (Tasks)
C1	Diagnosis
C2	Symptom Detection
C3	Hypothesis Generation
C4	Hypothesis Discrimination
C5	Generate Expectation
C6	Compare
C7	Classify
C8	Ask User
C9	Find Contributors
C10	Transform To Hypothesis Set
C11	Prediction Based Filtering
C12	Abstract
C13	Associate
C14	Probability Filter
C15	Select Hypothesis
C16	Collect Data
C17	Interpret Data
C18	Lookup
C19	Simulate
C20	Equal Check
C21	Determine Order of Magnitude
C22	Determine Ratio
C23	Check Against Threshold
C24	Teleological Abstract
C25	Statistical Compare
C26	Historical Compare
C27	Find Upstream
C28	Causal Covering
C29	Set Cover
C30	Intersect
C31	Subset Minimality Cover
C32	Cardinality Minimality Cover
C33	Select Random
C34	Suspend Constraint
C35	Check Consistency
C36	Delete
C37	Select Fault Model
C38	Estimate The Cost Of Testing The Hypothesis Set
C39	Order Hypothesis Set
C40	Select First
C41	Compiled Test
C42	Measure
C43	Deduce Input Vector
C44	Replace Hypothesis
C45	Split Hypothesis Set
C46	Obtain
C47	Estimate Local Cost
C48	Estimate Number Of Tests
C49	Estimate Overall Costs

Table B.1 - Diagnosis Tasks

Node ID number, Rectangle Nodes	Node Name (Methods)
R1	Prime Diagnostic
R2	Compare
R3	Classify
R4	User
R5	Model Based Hypothesis Generation
R6	Compiled
R7	Discrimination
R8	Lookup
R9	Simulate
R10	Exact
R11	Order Of Magnitude
R12	Threshold
R13	Teleological
R14	Statistical
R15	Historical
R16	Trace Back
R17	Causal Covering
R18	Prediction
R19	Set Cover
R20	Intersection
R21	Subset Minimality
R22	Cardinality Minimality
R23	Constraint Suspension
R24	Corroboration
R25	Fault Simulation
R26	Random
R27	Smart
R28	Compiled Test
R29	Probe
R30	Manipulate
R31	Replace
R32	Interpret In Isolation
R33	Split Interpret
R34	Based On Local Costs
R35	On The Number Of Tests
R36	Based On Overall Costs
R37	Direct Measure
R38	Indicator Based

Table B.2 - Diagnosis Methods

Node ID	Task/Method Name	Points to Node ID	Task/Method Name
C1	Diagnosis	R1	Prime Diagnostic
C2	Symptom Detection	R2, R3, R4	Compare, Classify, User
C3	Hypothesis Generation	R5, R6	Model-Based Hypothesis Generation, Compiled
C4	Hypothesis Discrimination	R7	Discrimination
C5	Generate Expectation	R8, R9	Lookup, Simulate
C6	Compare	R10, R11, R12, R13, R14, R15	Exact, Order of Magnitude, Threshold, Teleological, Statistical, Historical
C9	Find Contributors	R16, R17, R18	Trace Back, Causal Covering, Prediction
C10	Transform to Hypothesis Set	R19, R20, R21, R22	Set Cover, Intersection, Subset Minimality, Cardinality Minimality
C11	Prediction Based Filtering	R23, R24, R25	Constraint Suspension, Corroboration, Fault Simulation
C15	Select Hypothesis	R26, R27	Random, Smart
C16	Collect Data	R28, R29, R30, R31	Compiled Test, Probe, Manipulate, Replace
C17	Interpret Data	R32, R33, R5	Interpret in Isolation, Split Interpret, Model-Based Hypothesis Generation
C38	Estimate the Cost of Testing the Hypothesis Set	R34, R35, R36	Based on Local Costs, Based on the Number of Tests, Based on Overall Costs
C42	Measure	R37, R38	Direct Measure, Indicator Based
R1	Prime Diagnostic	C2, C3, C4.	Symptom Detection, Hypothesis Generation, Hypothesis Generation.
R2	Compare	C5, C6	Generate Expectation, Compare
R3	Classify	C7	Classify
R4	User	C8	Ask User
R5	Model-Based Hypothesis Generation	C9, C10, C11	Find Contributors, Transform to Hypothesis Set, Prediction Based Filtering
R6	Compiled	C12, C13, C14	Abstract, Associate, Probability Filter
R7	Discrimination	C15, C16, C17	Select Hypothesis, Collect Data, Interpret Data
R8	Lookup	C18	Lookup
R9	Simulate	C19	Simulate
R10	Exact	C20	Equal Check
R11	Order of Magnitude	C21, C20	Determine Order of Magnitude, Equal Check
R12	Threshold	C22, C23	Determine Ratio, Check Against Threshold
R13	Teleological	C24, C20	Teleological Abstract, Equal Check
R14	Statistical	C25	Statistical Compare
R15	Historical	C26	Historical Compare

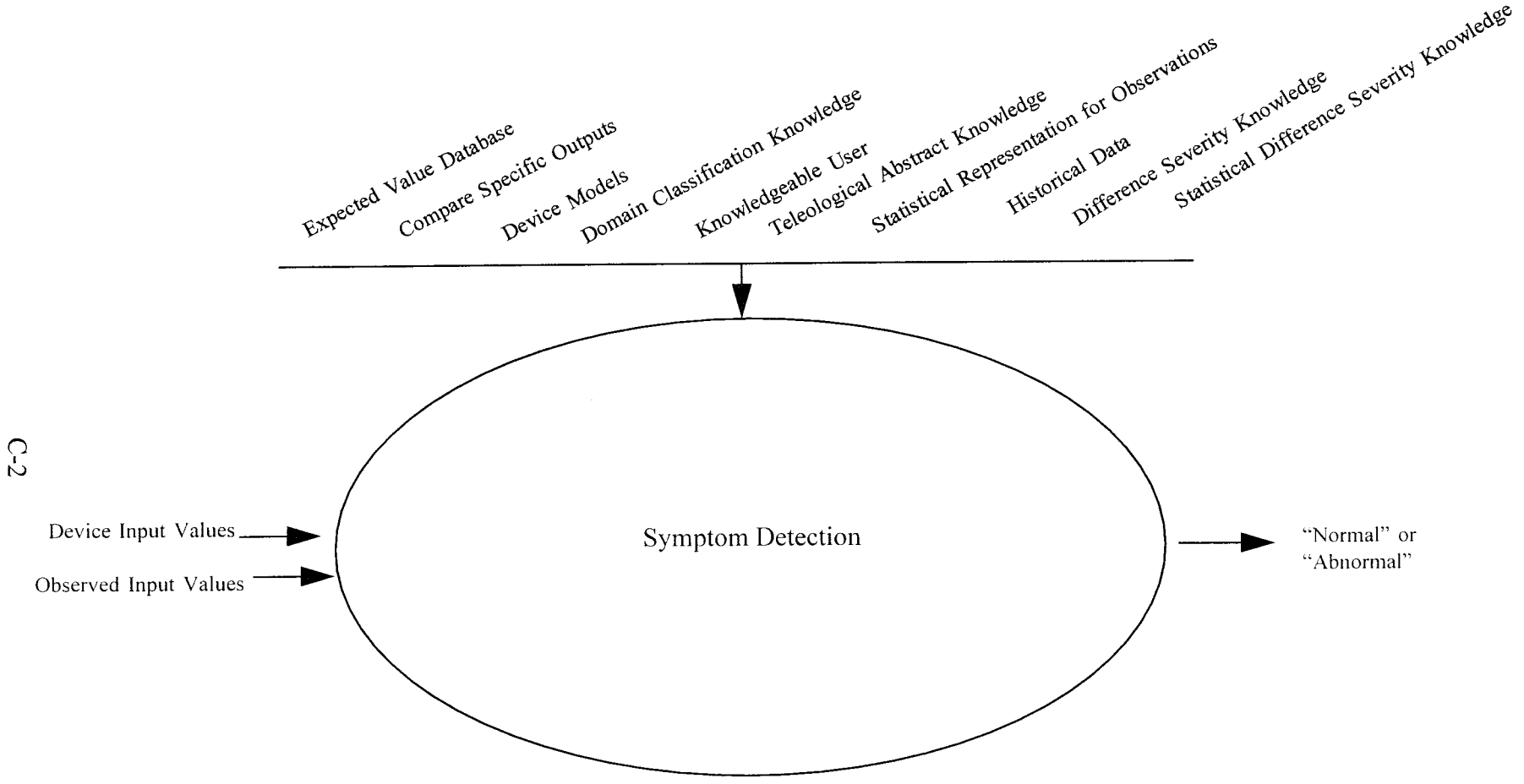
Table B.3 - Task Structure Links

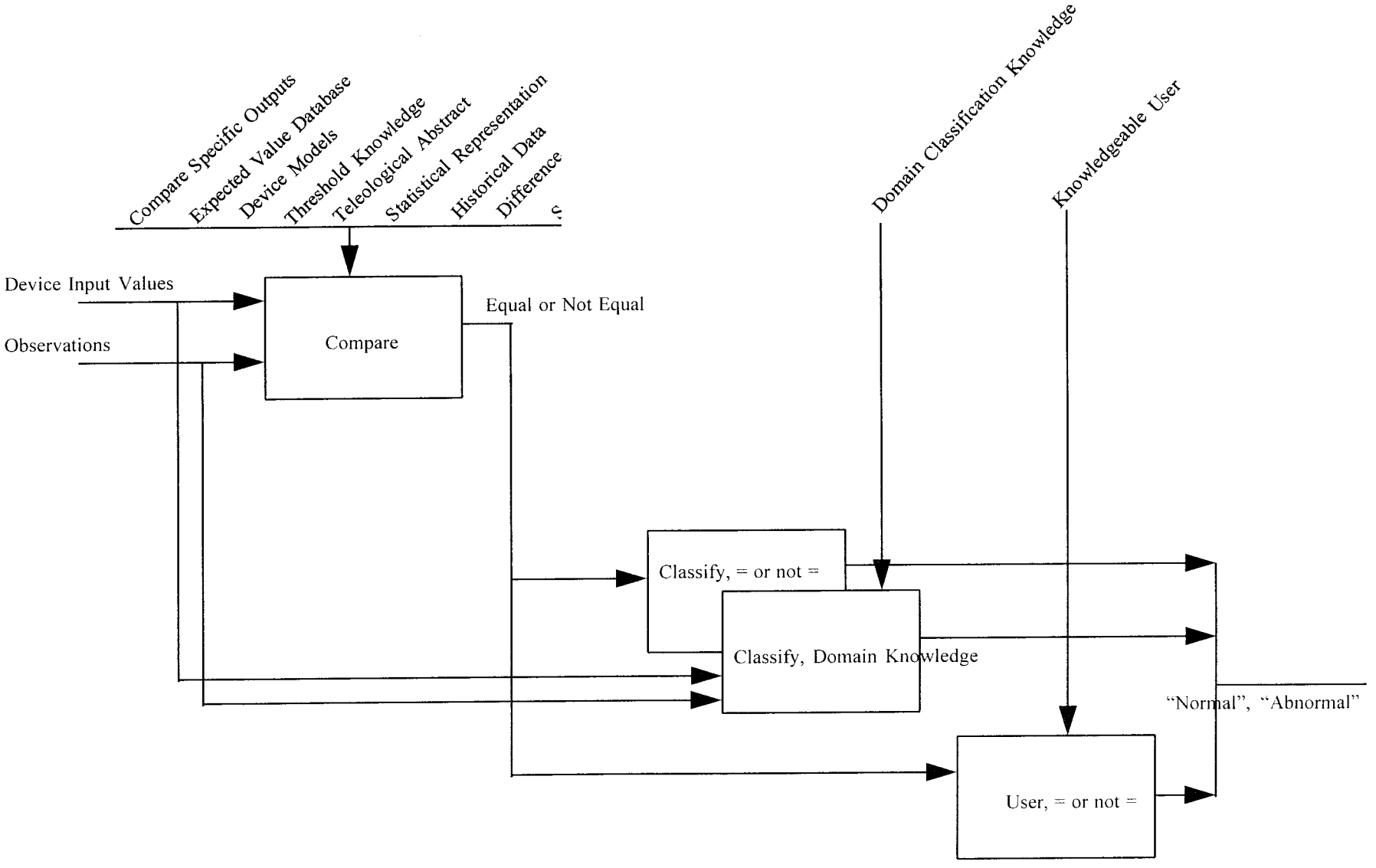
R16	Trace Back	C27	Find Upstream
R17	Causal Covering	C28	Causal Covering
R18	Prediction	C19	Simulate
R19	Set Cover	C29	Set Cover
R20	Intersection	C30	Intersect
R21	Subset Minimality	C31	Subset Minimality Cover
R22	Cardinality Minimality	C32	Cardinality Minimality Cover
R23	Constraint Suspension	C33, C34, C19, C35, C36	Select Random, Suspend Constraint, Simulate, Check Consistency, Delete
R24	Corroboration	C33, C19, C6, C36	Select Random, Simulate, Compare, Delete
R25	Fault Simulation	C33, C37, C19, C6, C36	Select Random, Select Fault Model, Simulate, Compare, Delete
R26	Random	C33	Select Random
R27	Smart	C38, C39, C40	Estimate the Cost of Testing the Hypothesis Set, Order Hypothesis Set, Select First
R28	Compiled Test	C41	Compiled Test
R29	Probe	C42, C5, C6	Measure, Generate Expectation, Compare
R30	Manipulate	C43, C19, C46, C6	Deduce Input Vector, Simulate, Obtain, Compare
R31	Replace	C44, C5, C6	Replace Hypothesis, Generate Expectation, Compare
R32	Interpret in Isolation	C36	Delete
R33	Split Interpret	C45	Split Hypothesis Set
R34	Based on Local Costs	C47	Estimate Local Costs
R35	Based on the Number of Tests	C48	Estimate Number of Tests
R36	Based on Overall Costs	C49	Estimate Overall Costs
R37	Direct Measure	C46	Obtain

Table B.3 - Task Structure Links (Continued)

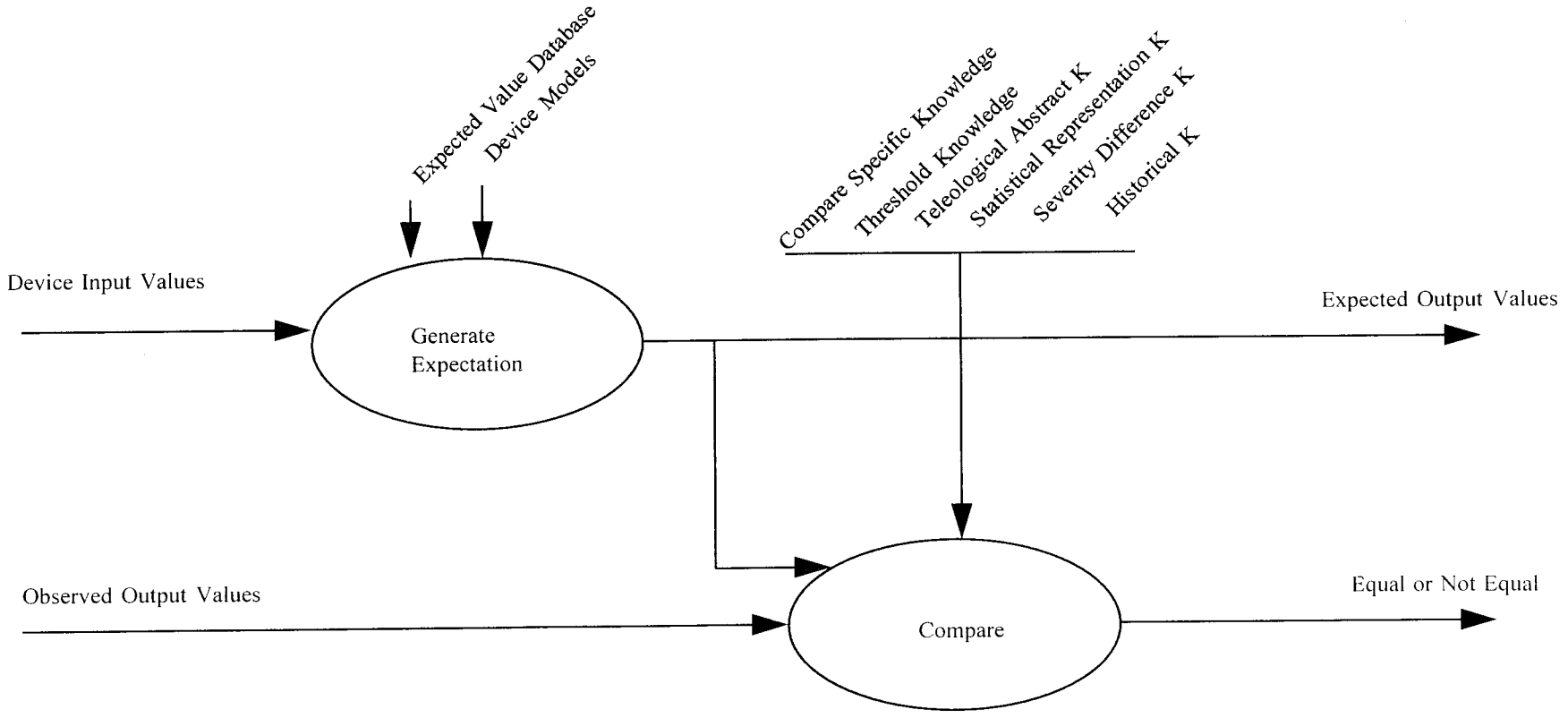
APPENDIX C. Modified IDEF-0 Task Structure Analysis

This appendix contains the modified IDEF-0 model for the diagnosis task structure (see Appendix B) used in GTS. IDEF-0 is an industrial engineering approach to modeling processes. Starting at the highest level functions for a process, an IDEF-0 model decomposes a process into the lower level functions and sub-functions that realize it. For each function, the IDEF-0 model identifies the inputs, outputs, controls and mechanisms which make the function operate. Applied to the diagnosis task structure, for each task or method, the modified IDEF-0 model identifies the task's or method's input, output, and knowledge requirements. Applying the model consecutively up the task structure collects the input, output, and knowledge requirements of lower levels into upper levels. The input, output, and knowledge requirements for a task are the collection of the input, output, and knowledge requirements of the tasks and methods organized below it. The requirements for the overall diagnosis process is then the collection of input, output, and knowledge requirements of the tasks and methods contained in the entire task structure. The knowledge requirements obtained from the IDEF-0 modeling analysis are what are placed in the sponsor for each diagnosis problem solving method (Chapter 4, section 2.2.3). Furthermore, the input and output requirements serve as a basis for specifying the sequencing knowledge needed when the control section decides what problem solving method to choose (Chapter 4, section 2.3, Chapter 5, section 3). This is so because the IDEF-0 models show the data flow interconnections between the problem solving methods. The IDEF-0 model is developed by traversing the task structure from top to bottom and from left to right. Therefore, the first diagram is for the Symptom Detection task, then all the tasks and methods organized under it, then the tasks and methods under the Hypothesis Generation task are modeled, and finally the same is done for the Hypothesis Discrimination task.

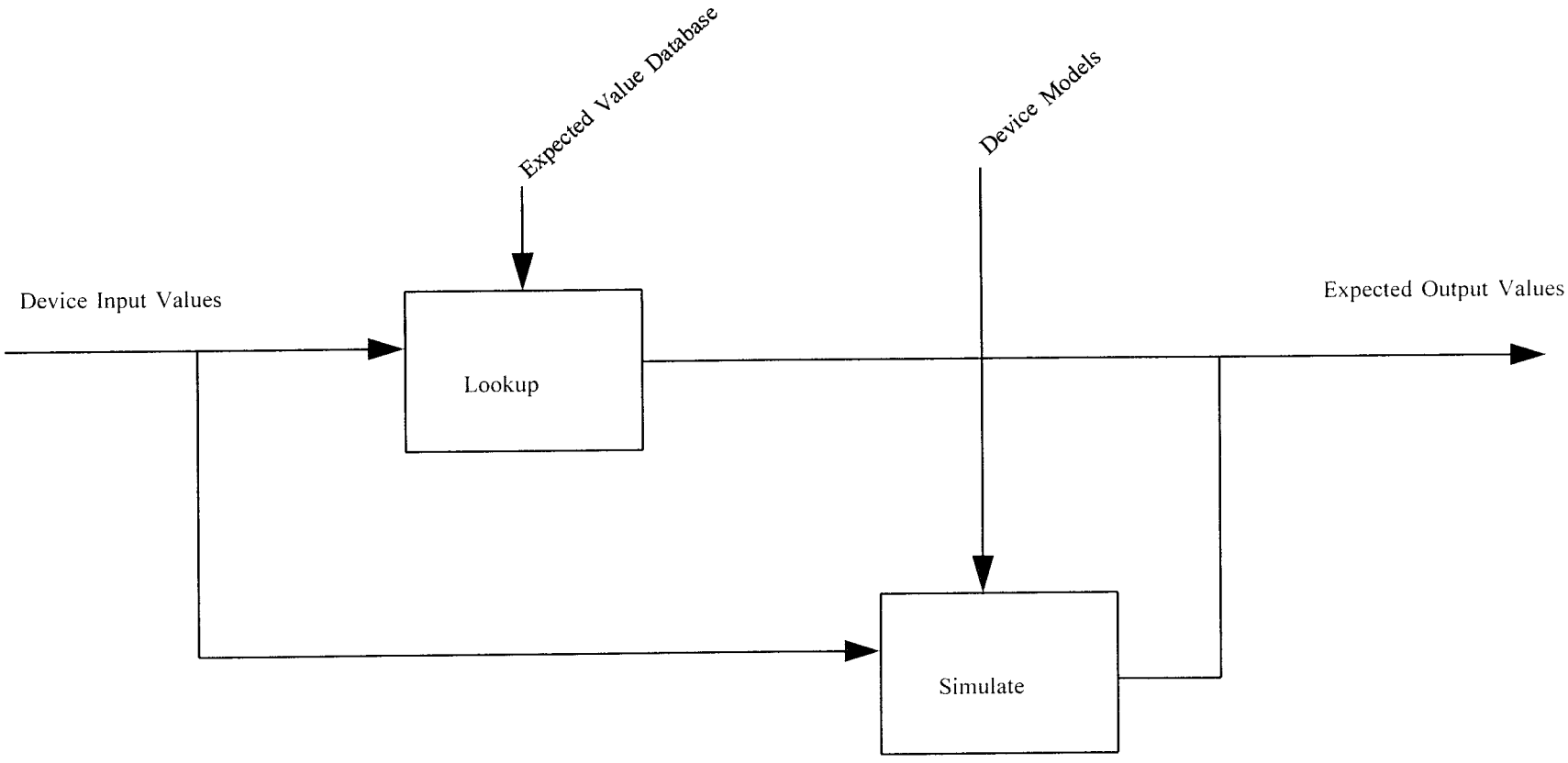

Expected Value Database
Compare Specific Outputs
Device Models
Domain Classification Knowledge
Knowledgeable User
Teleological Abstract Knowledge
Statistical Representation for Observations
Historical Data
Difference Severity Knowledge
Statistical Difference Severity Knowledge
Device Input Values
Observed Input Values
Symptom Detection
"Normal" or
"Abnormal"



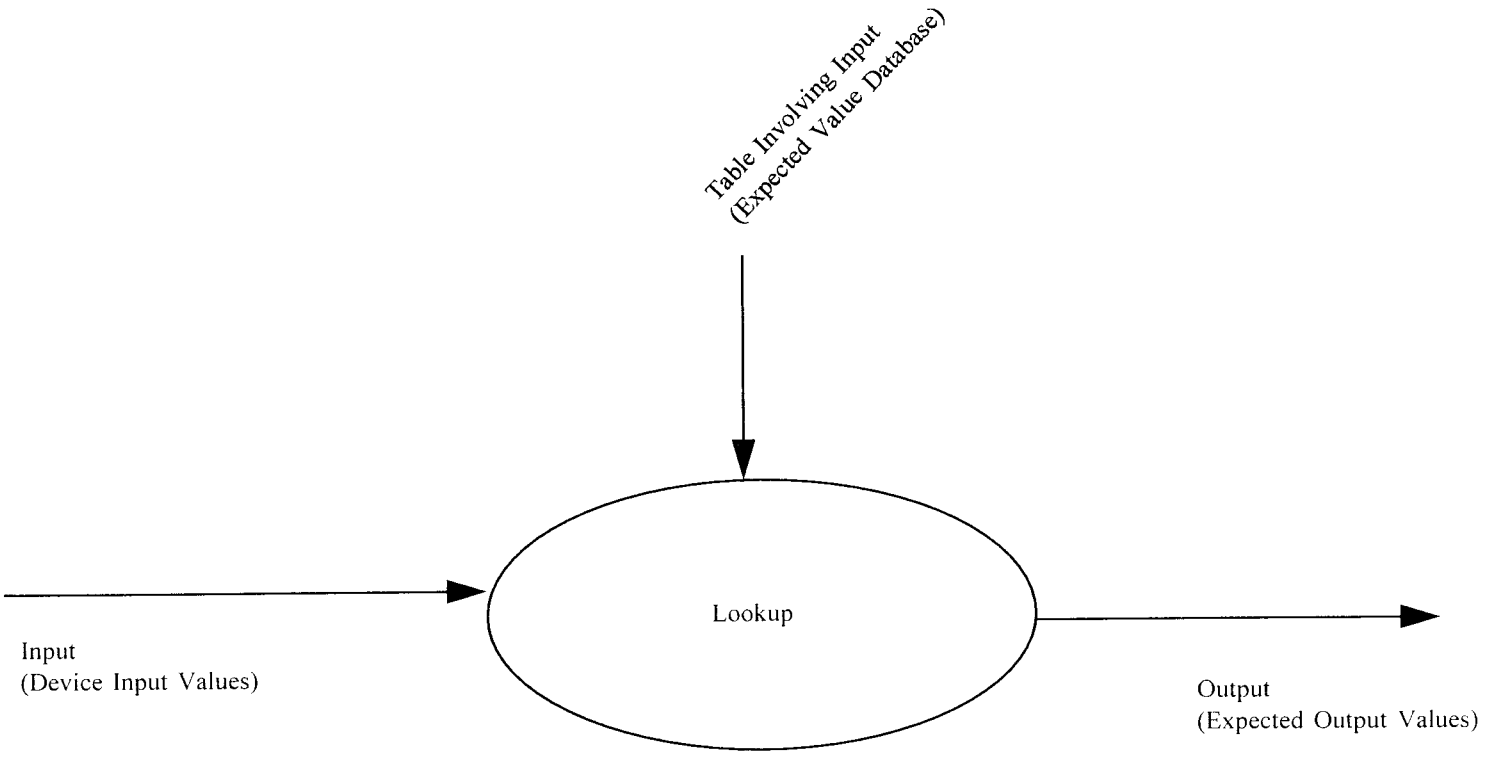
Symptom Detection



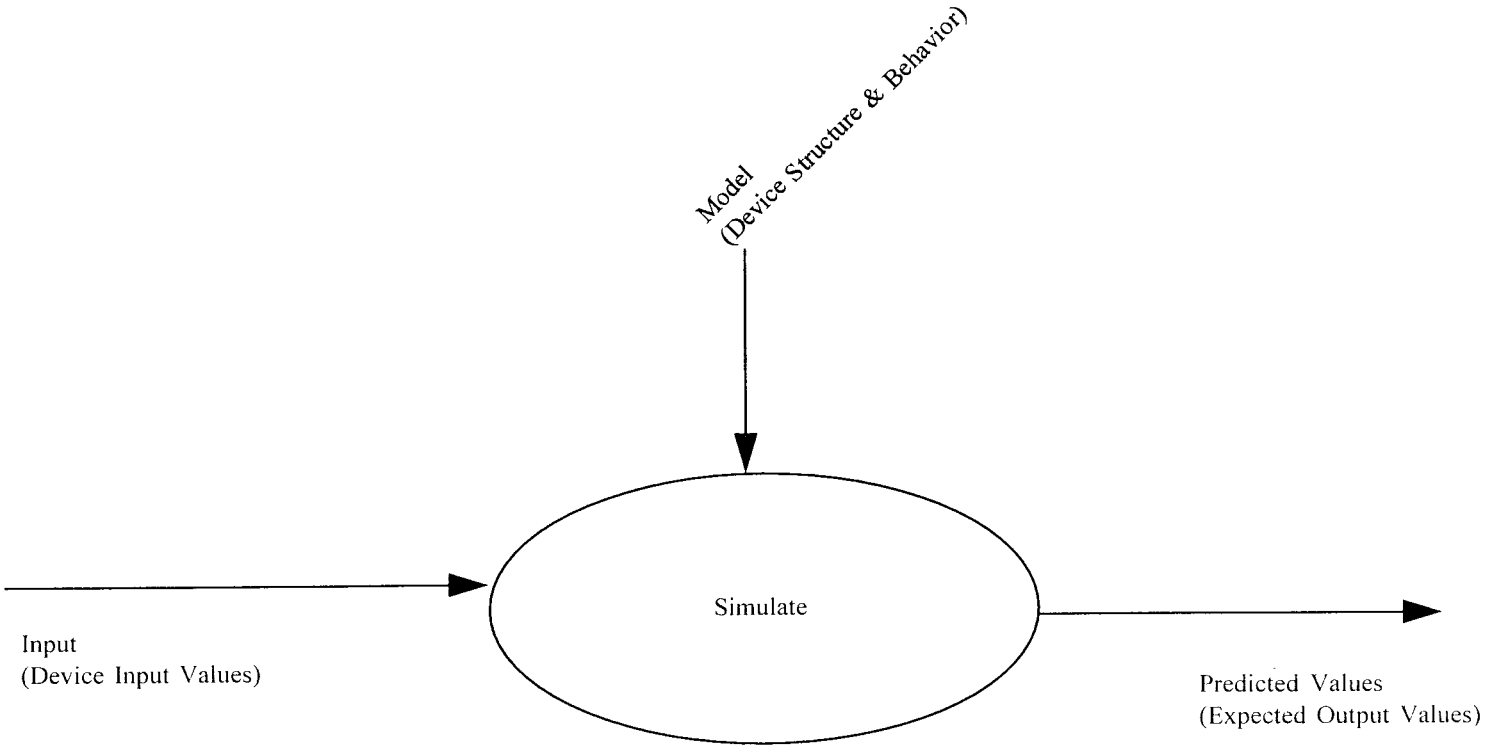
Compare Method



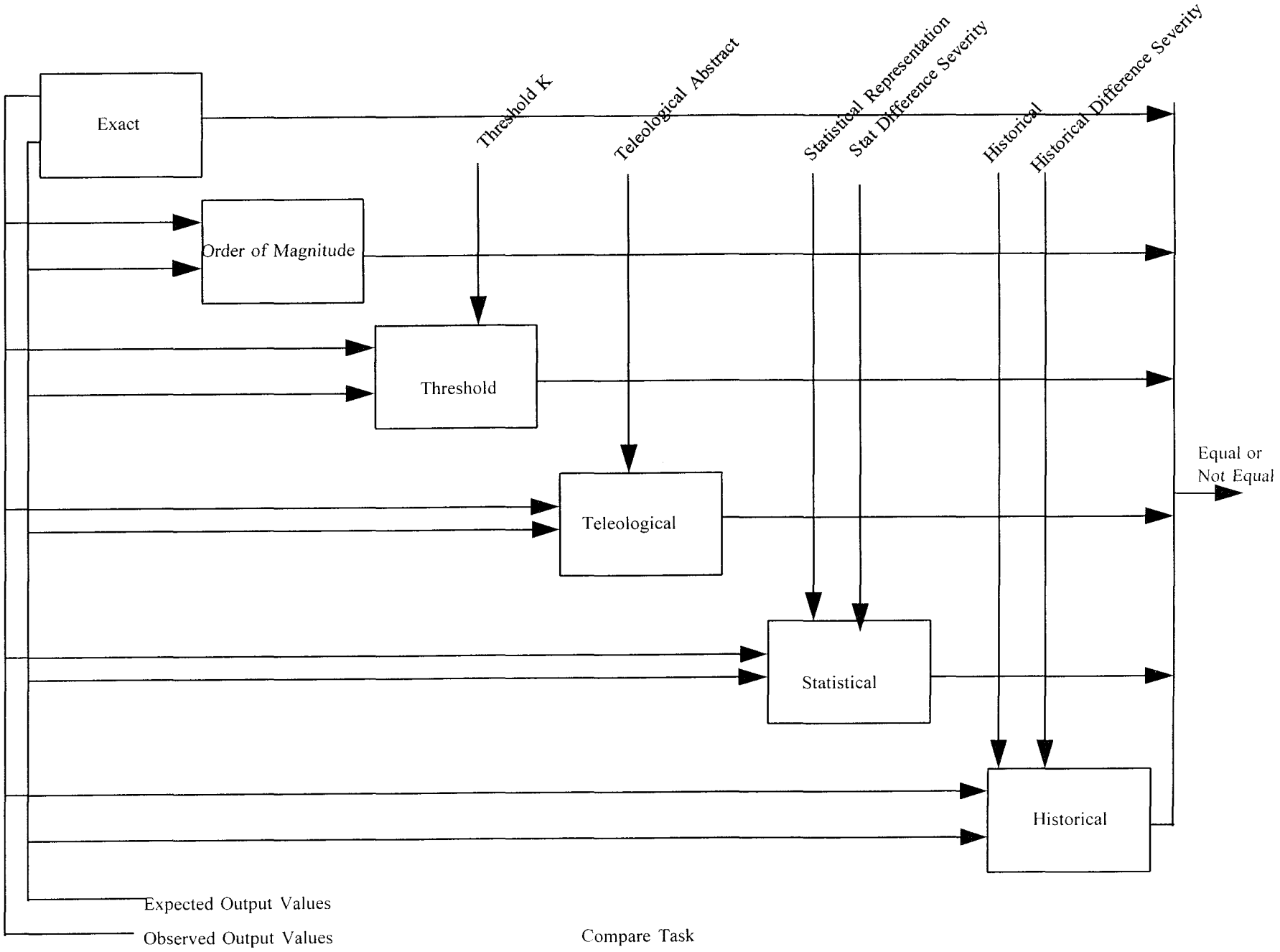
Generate Expectation

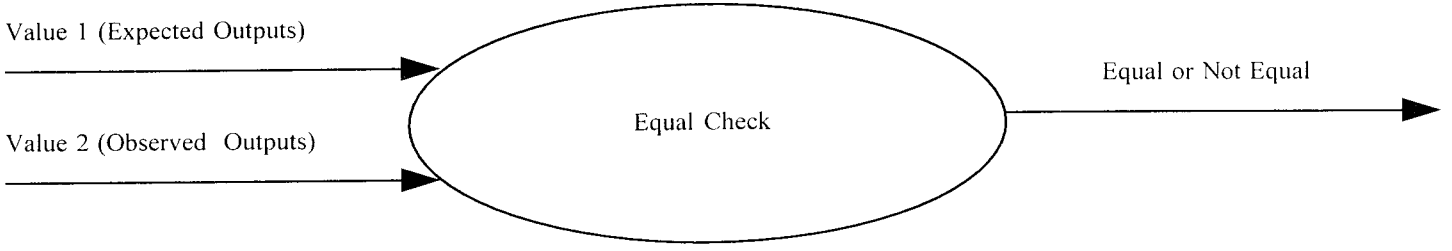


Lookup Primitive Inference

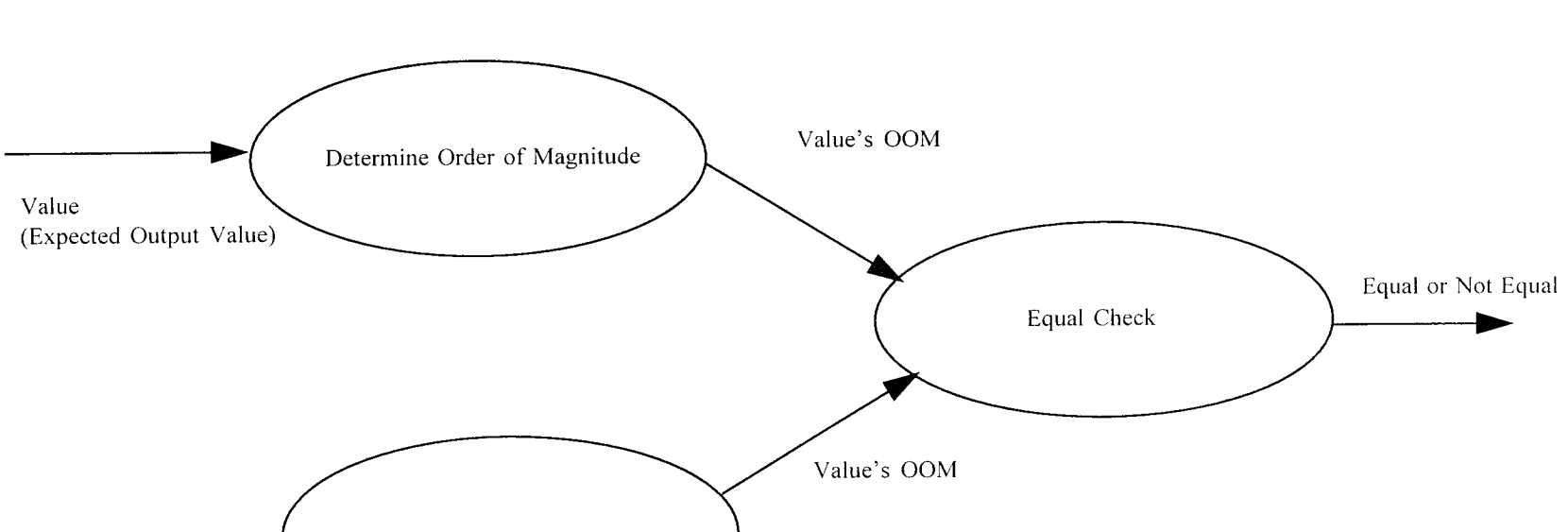


Simulate Primitive Inference

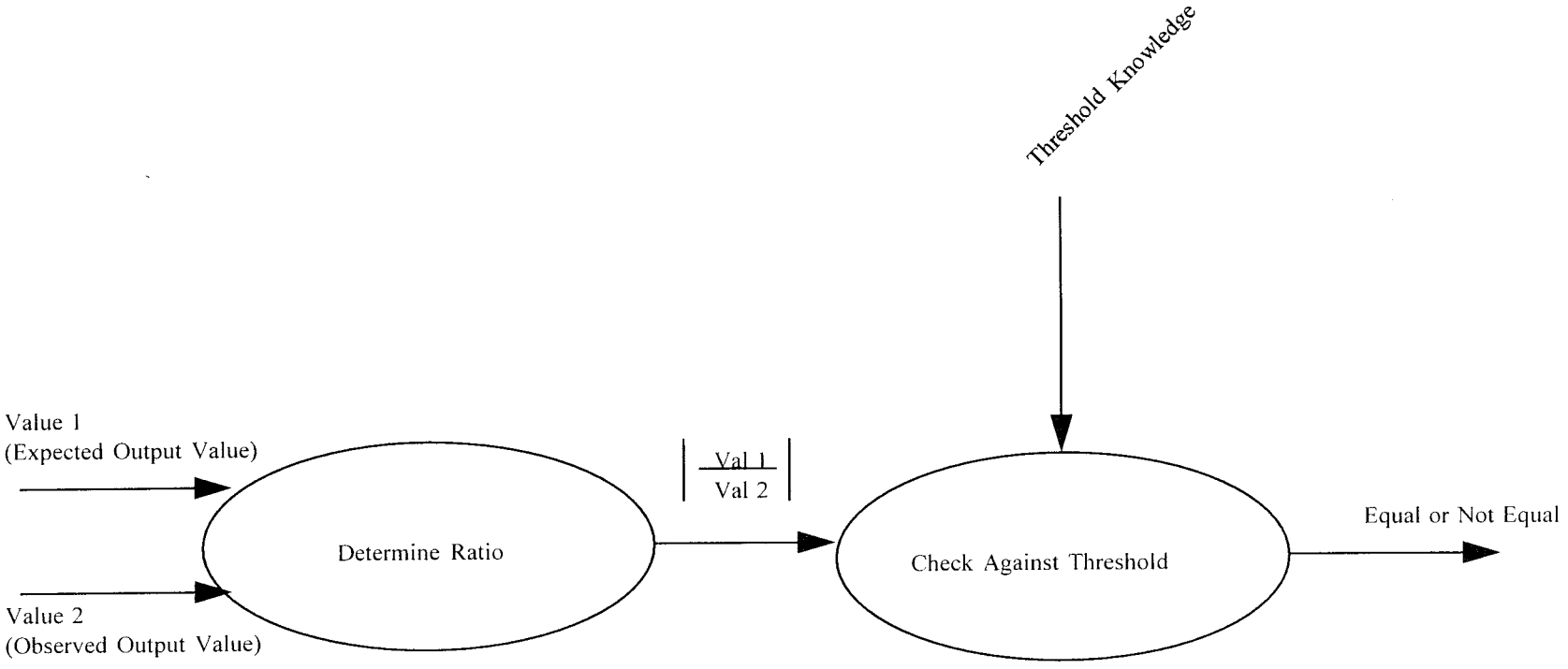

Exact
Order of Magnitude
Threshold
Teleological
Statistical
Historical
Threshold K
Teleological Abstract
Statistical Representation
Stat Difference Severity
Historical
Historical Difference Severity
Equal or
Not Equal
Expected Output Values
Observed Output Values
Compare Task


Value 1 (Expected Outputs)
Value 2 (Observed Outputs)
Equal Check
Equal or Not Equal

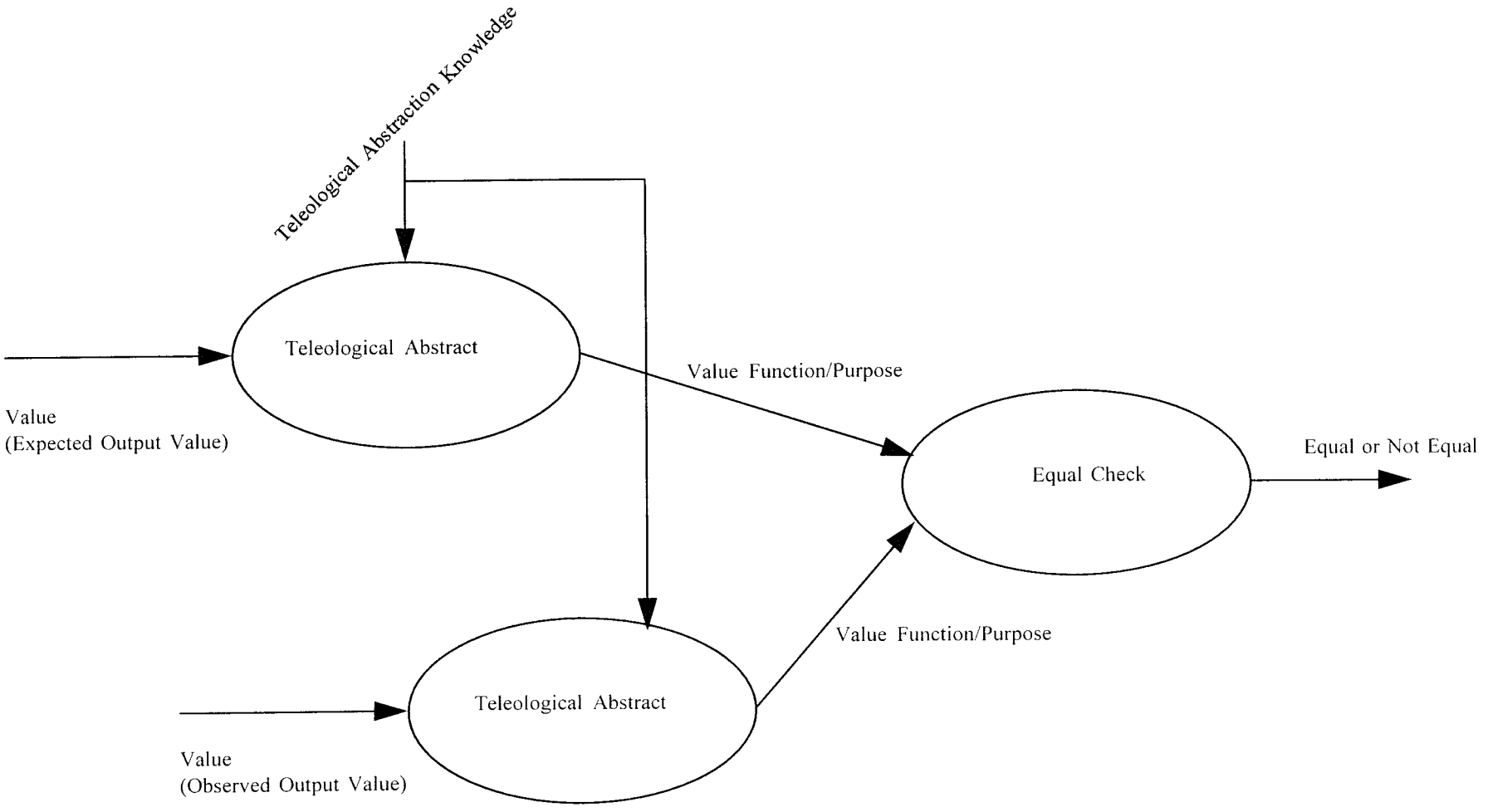
Exact



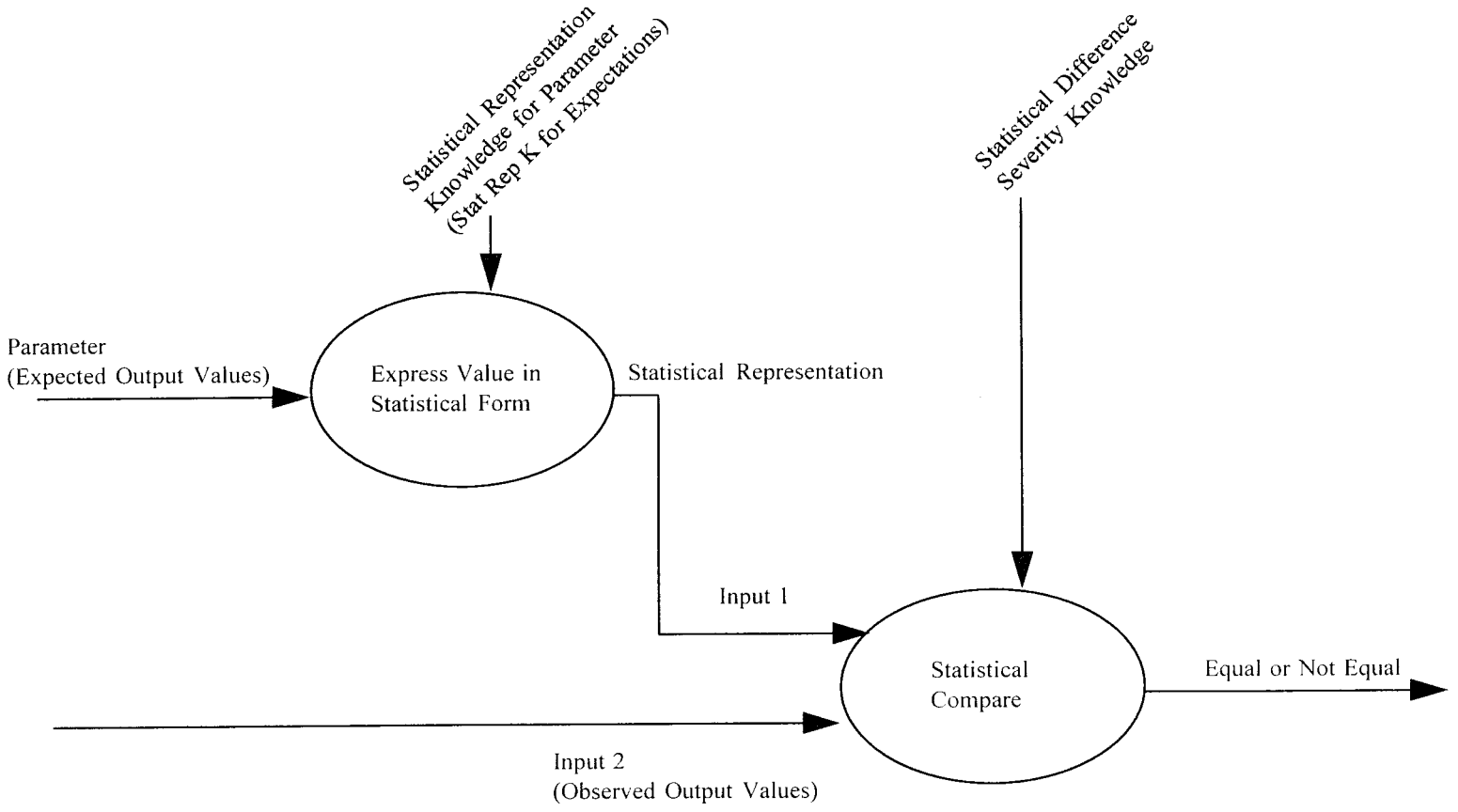
Order of Magnitude


Threshold Knowledge
Value 1
(Expected Output Value)
Value 2
(Observed Output Value)
Determine Ratio
Val 1
Val 2
Check Against Threshold
Equal or Not Equal

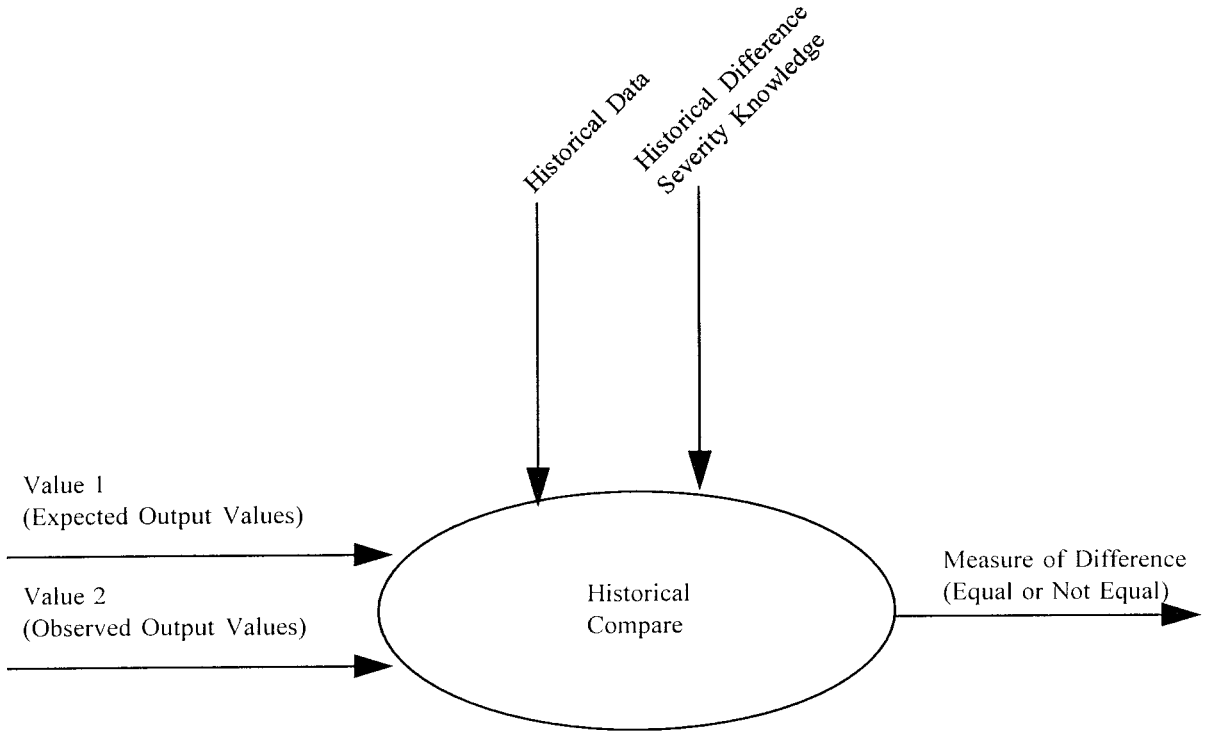
Threshold



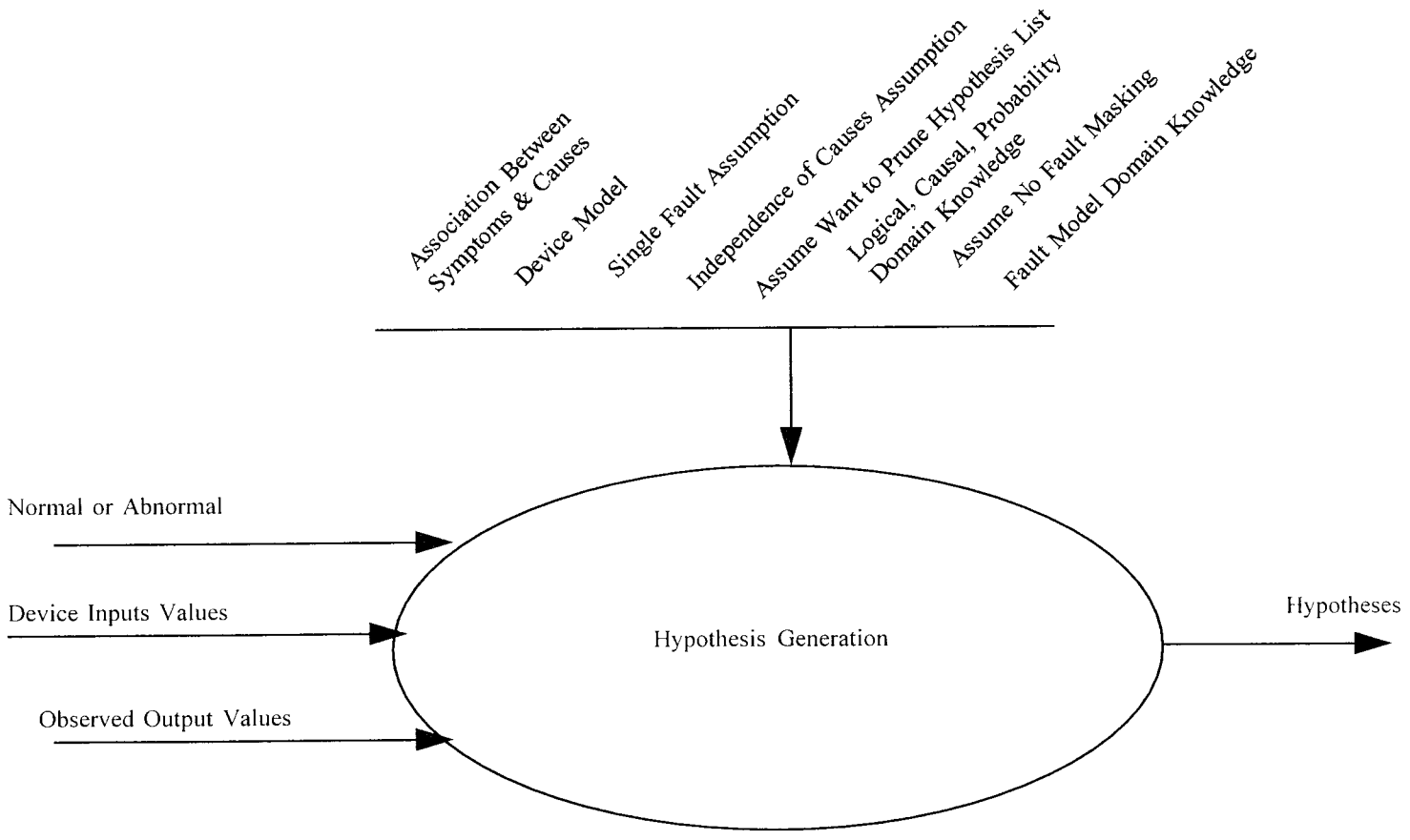
Teleological Method

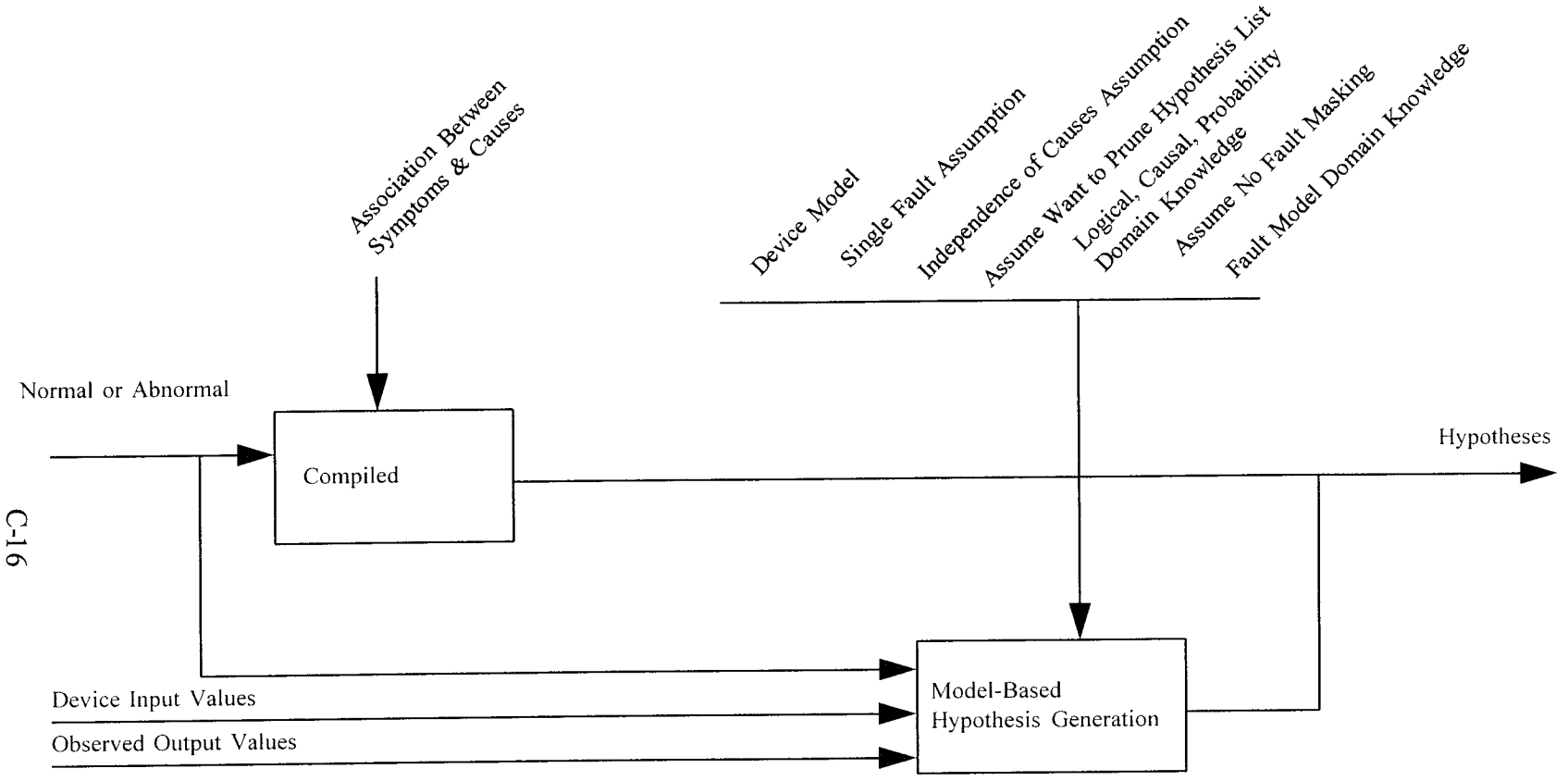


Statistical Method

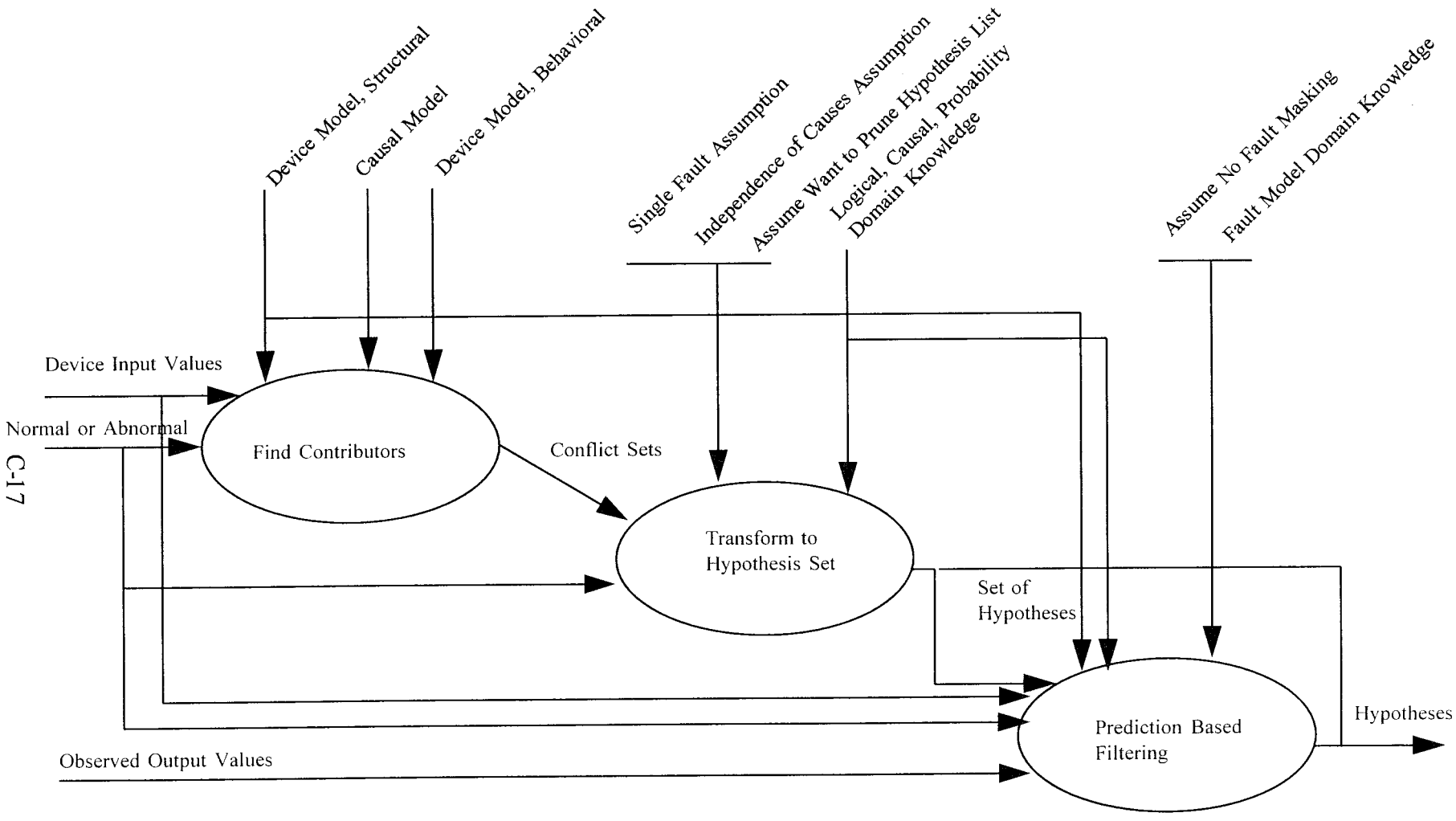


Historical Compare Method

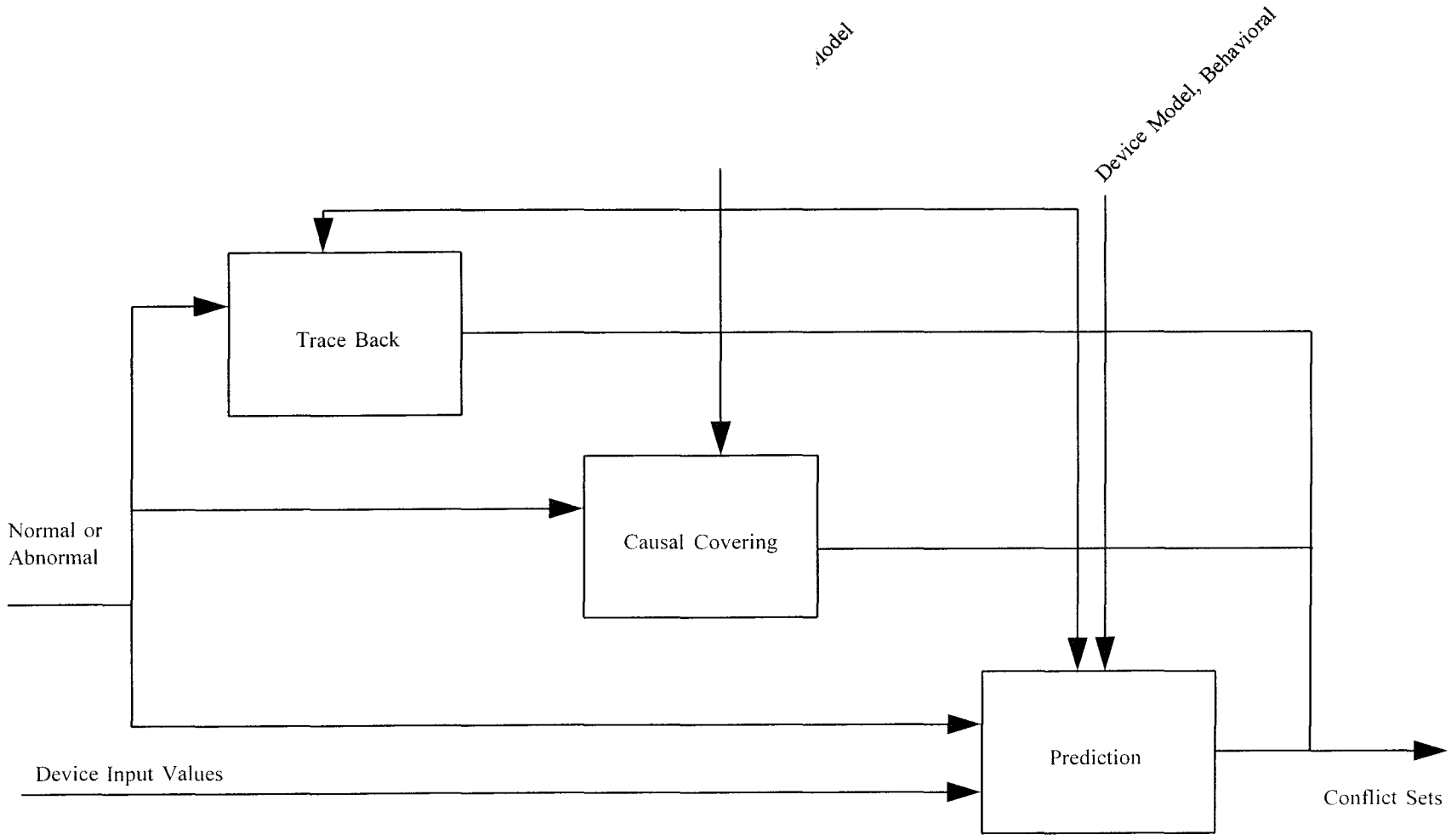

Association Between Symptoms & Causes
Device Model
Single Fault Assumption
Independence of Causes Assumption
Assume Want to Prune Hypothesis List
Logical, Causal, Probability
Domain Knowledge
Assume No Fault Masking
Fault Model Domain Knowledge
Normal or Abnormal
Device Inputs Values
Observed Output Values
Hypothesis Generation
Hypotheses



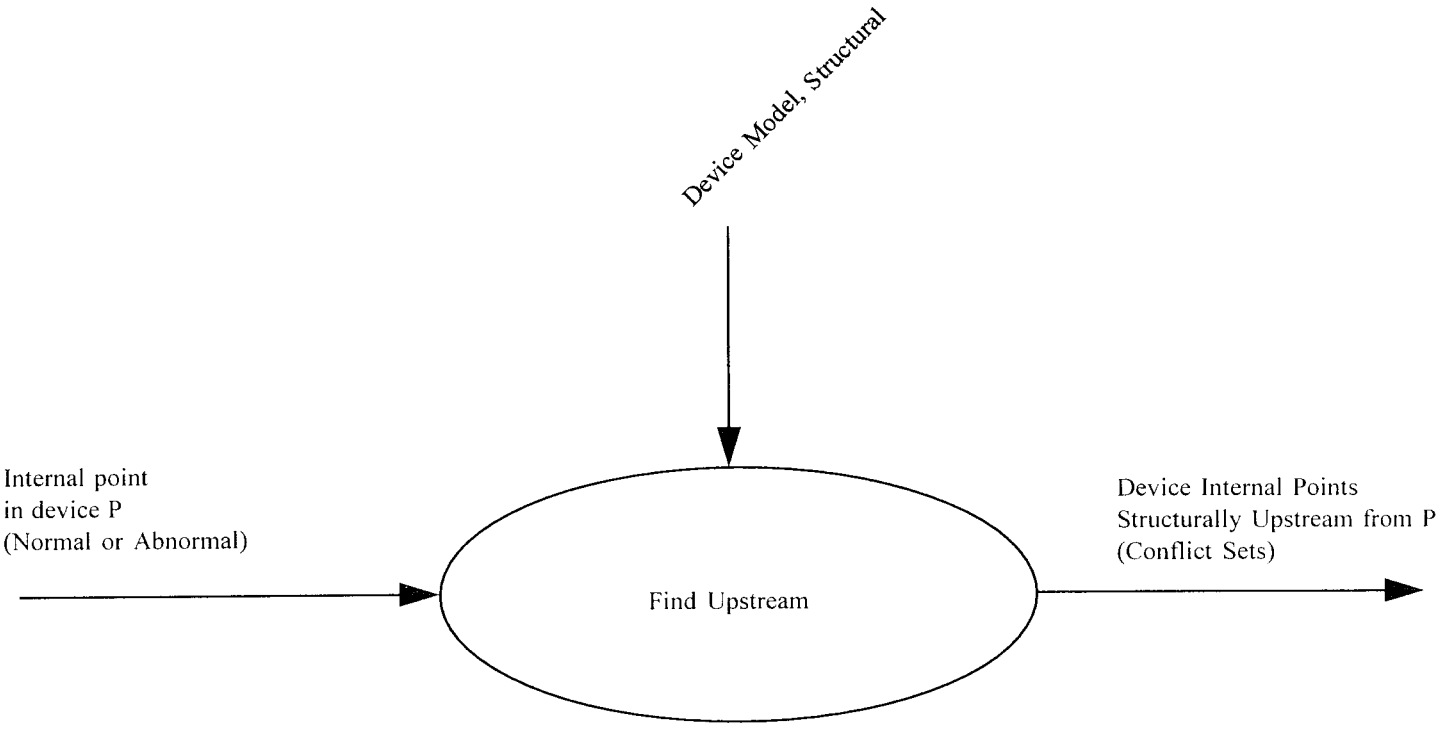
Hypothesis Generation



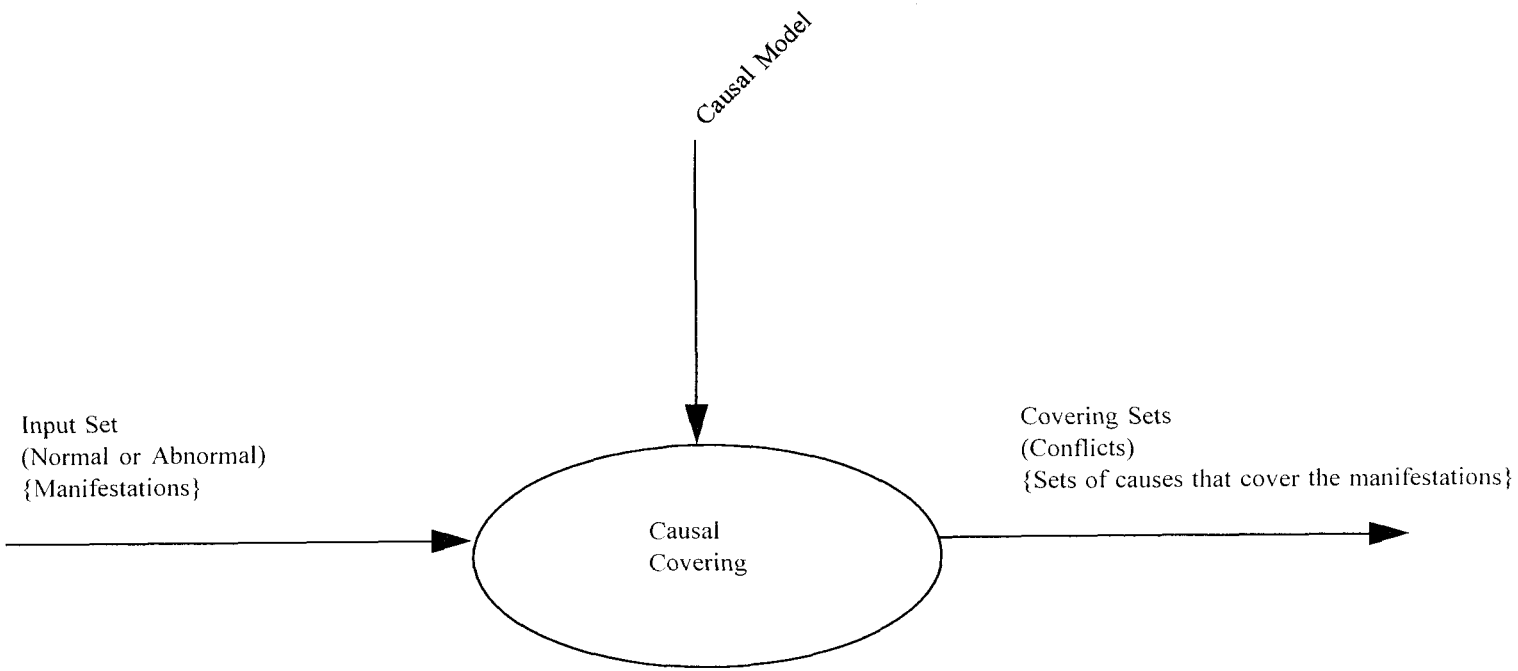
Model-Based Hypothesis Generation



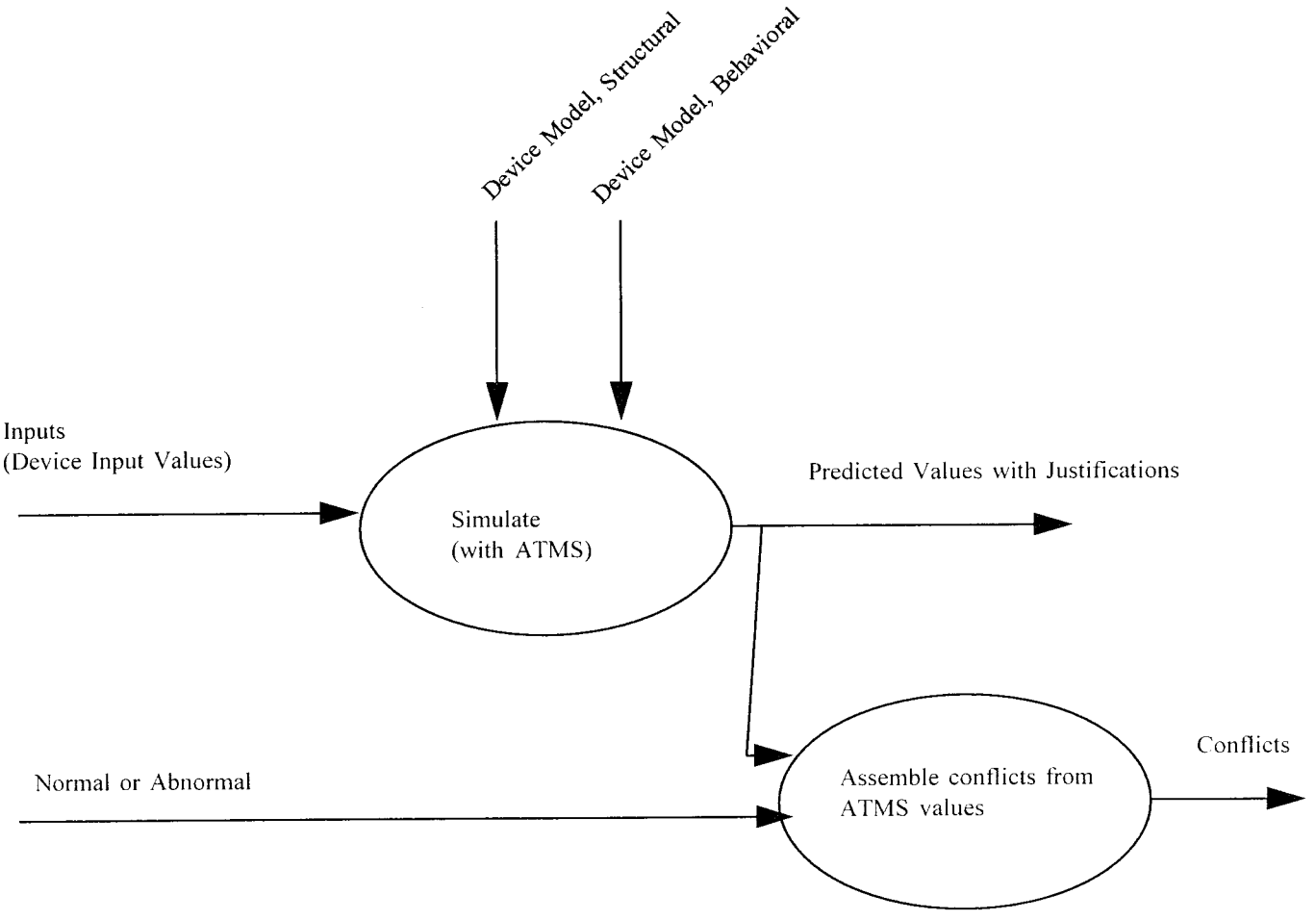
Find Contributors


Device Model, Structural
Internal point
in device P
(Normal or Abnormal)
Find Upstream
Device Internal Points
Structurally Upstream from P
(Conflict Sets)

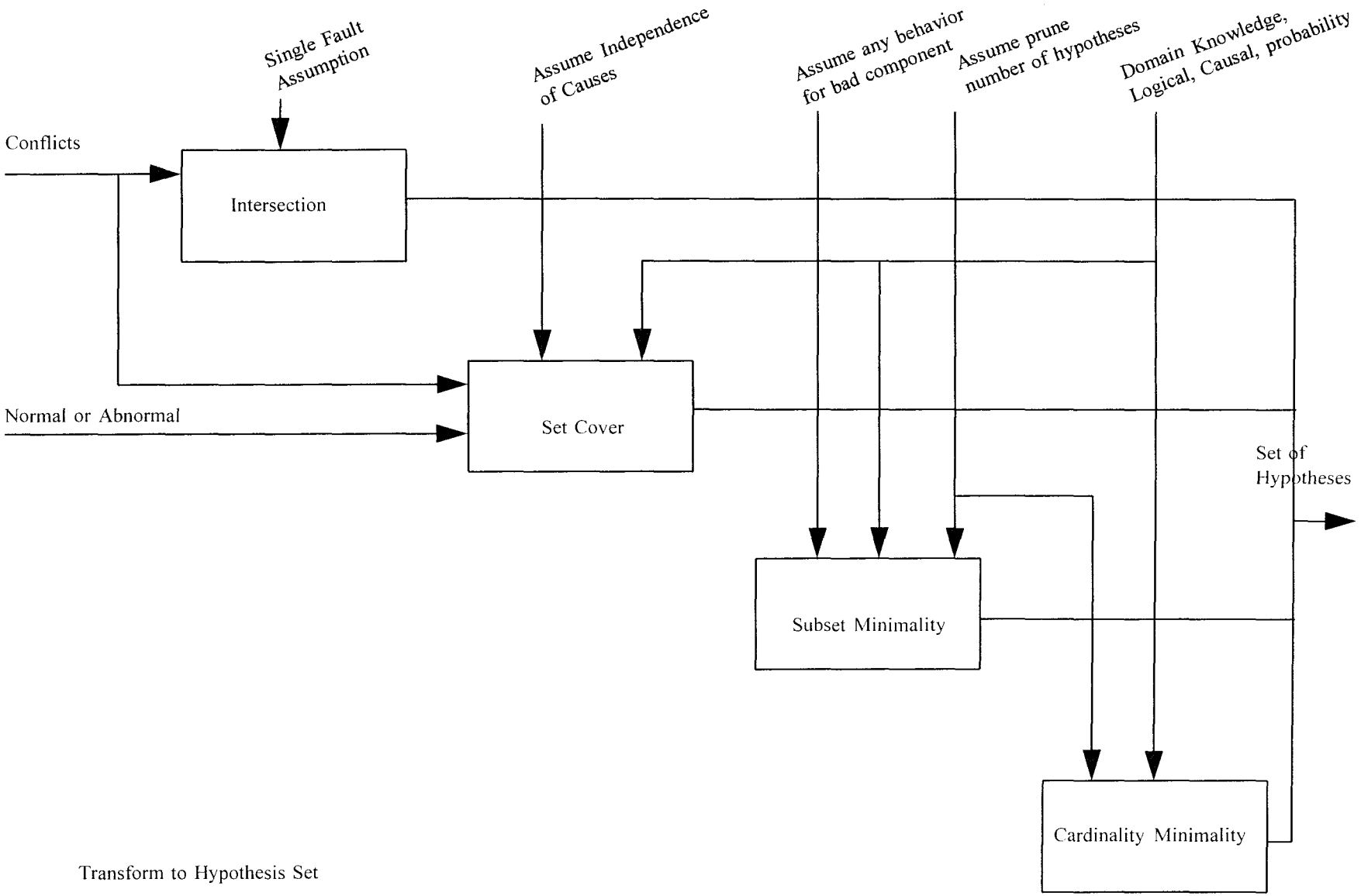
Trace Back



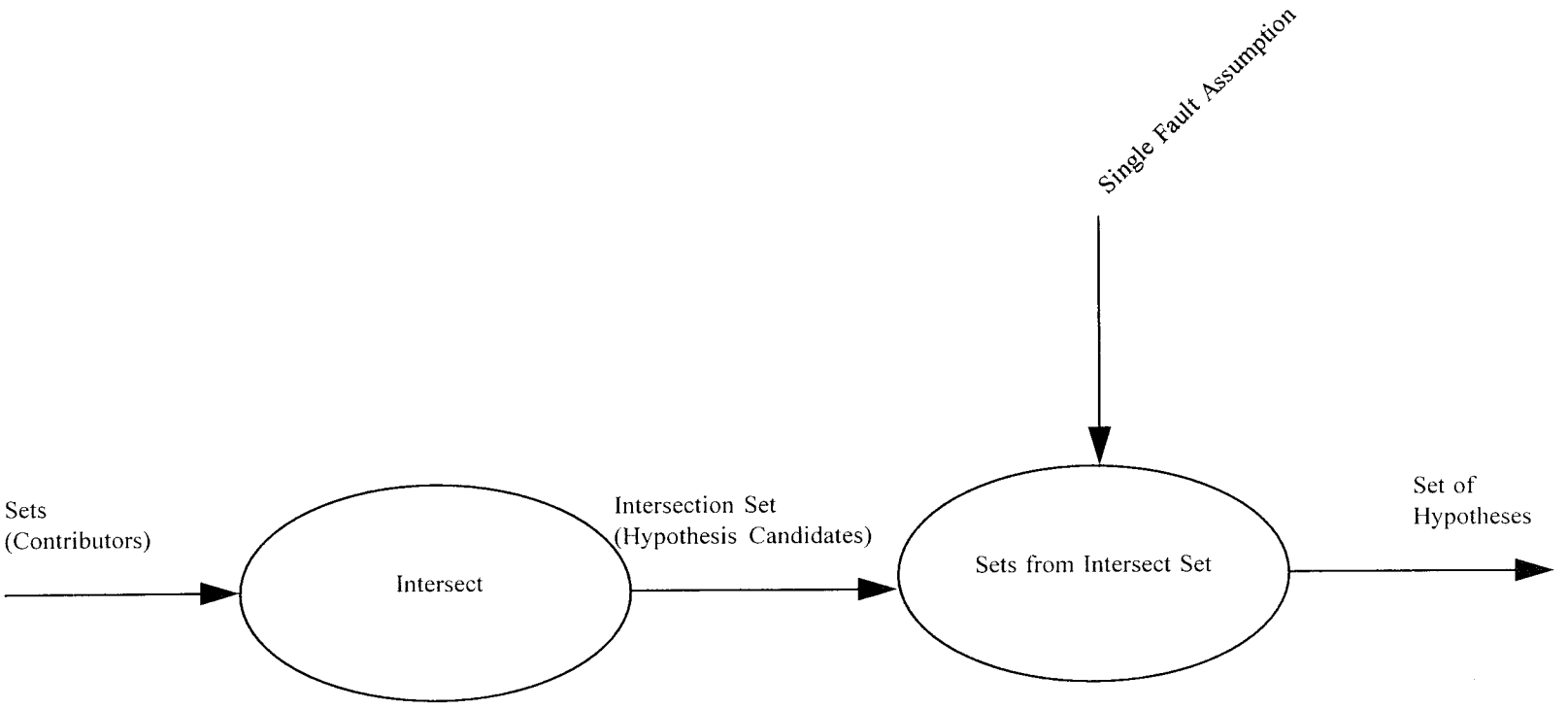
Causal Covering


Device Model, Structural
Device Model, Behavioral
Inputs
(Device Input Values)
Simulate
(with ATMS)
Predicted Values with Justifications
Normal or Abnormal
Assemble conflicts from
ATMS values
Conflicts

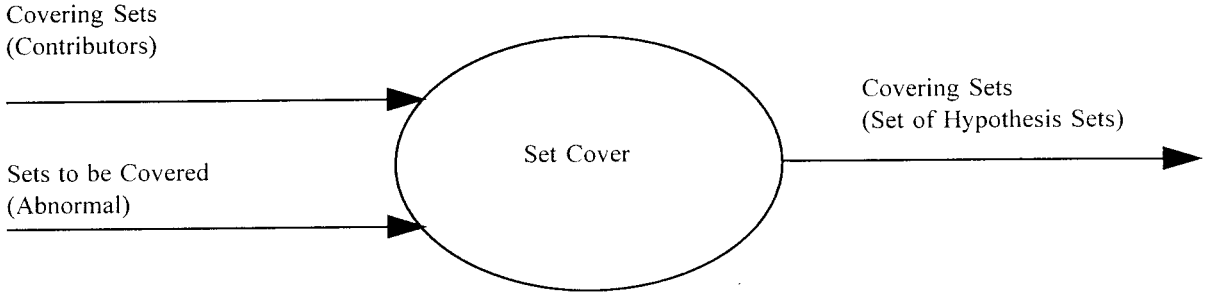
Prediction



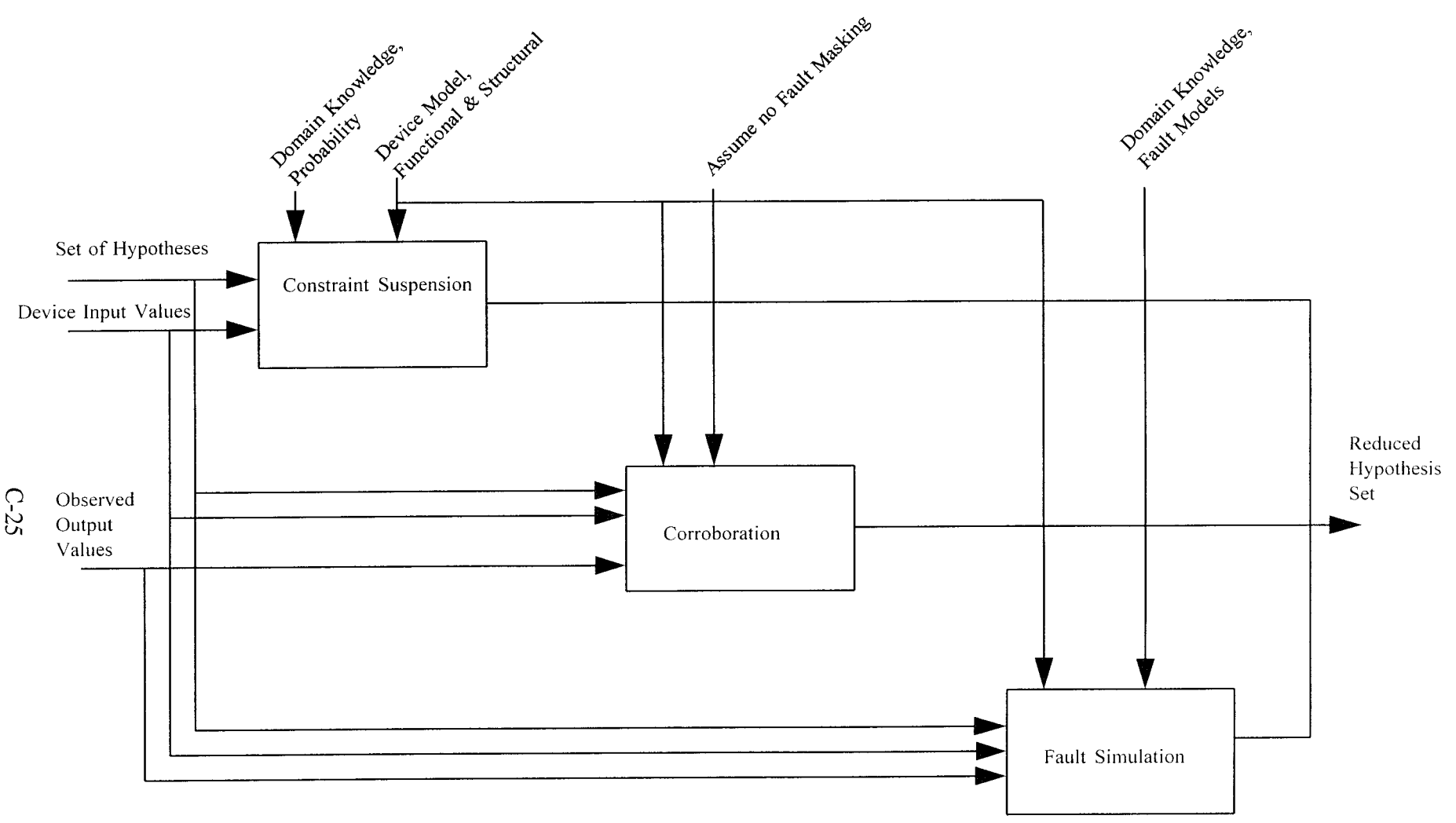
Transform to Hypothesis Set



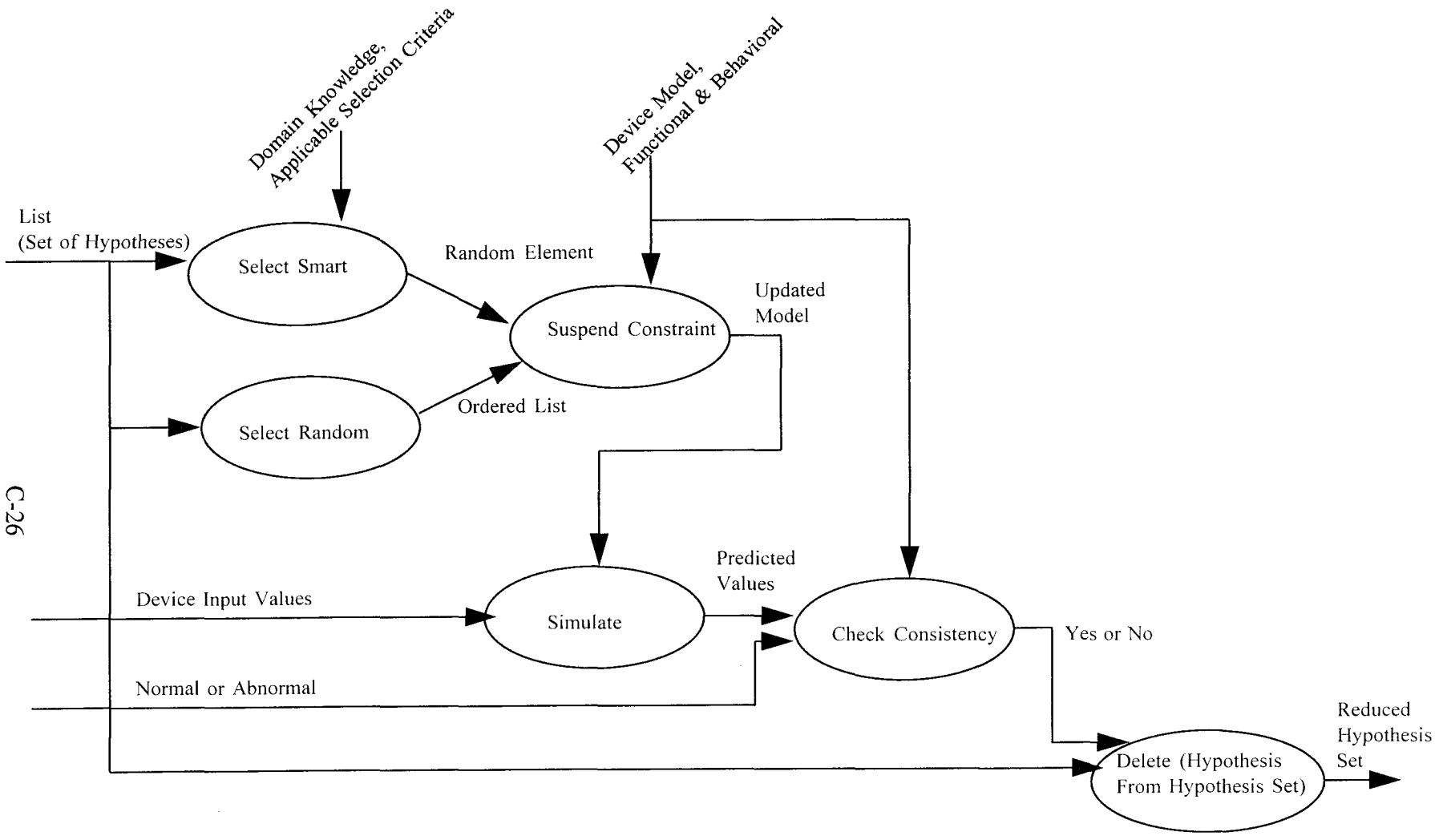
Intersection


Covering Sets
(Contributors)
Sets to be Covered
(Abnormal)
Set Cover
Covering Sets
(Set of Hypothesis Sets)

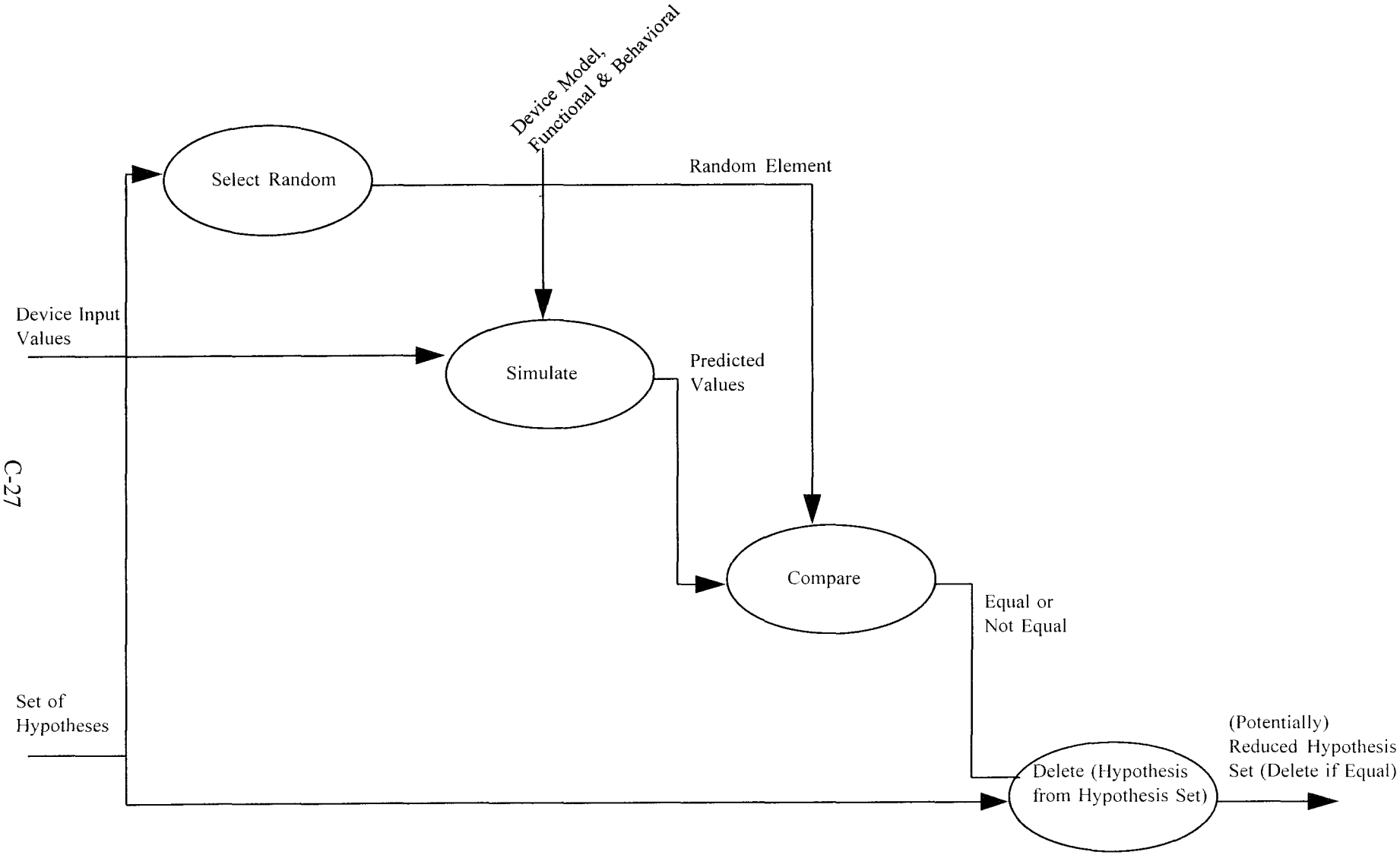
Set Cover



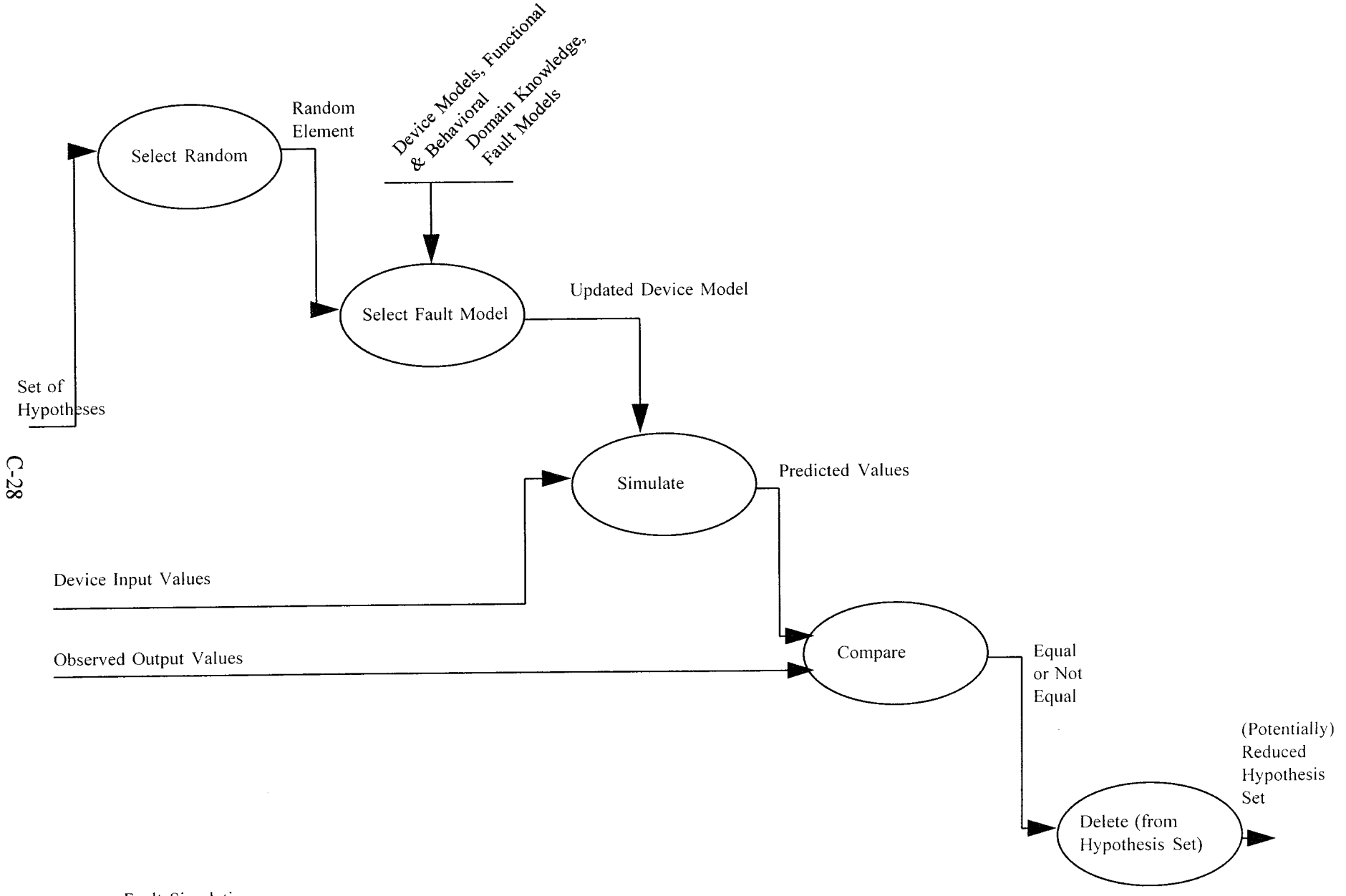
Prediction-Based Filtering



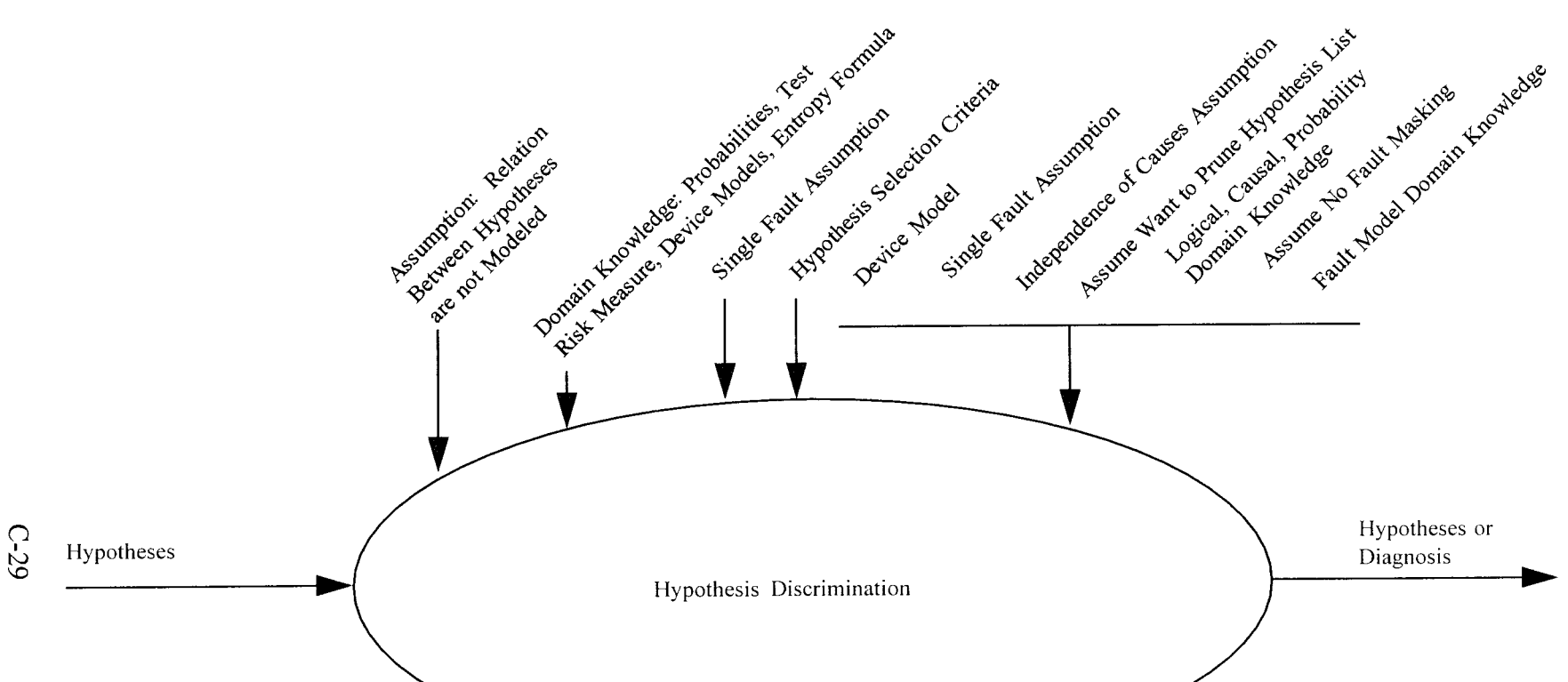
Constraint Suspension

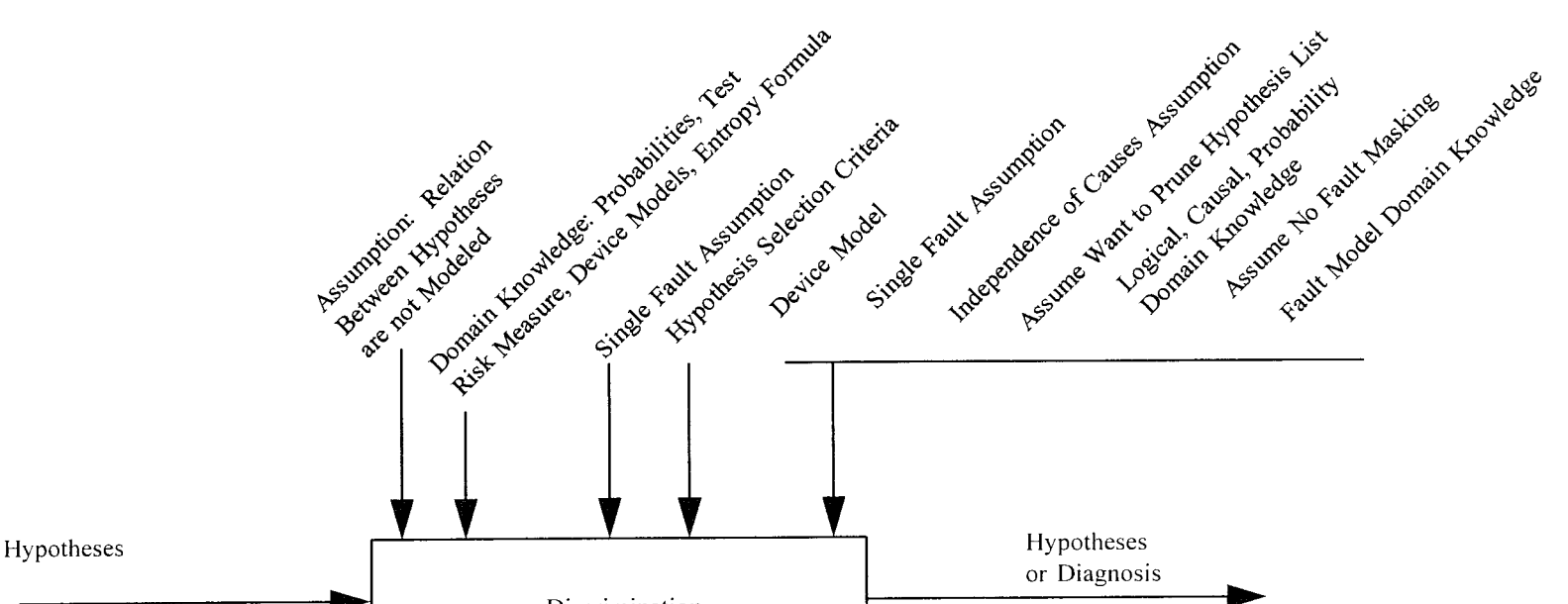


Corroboration

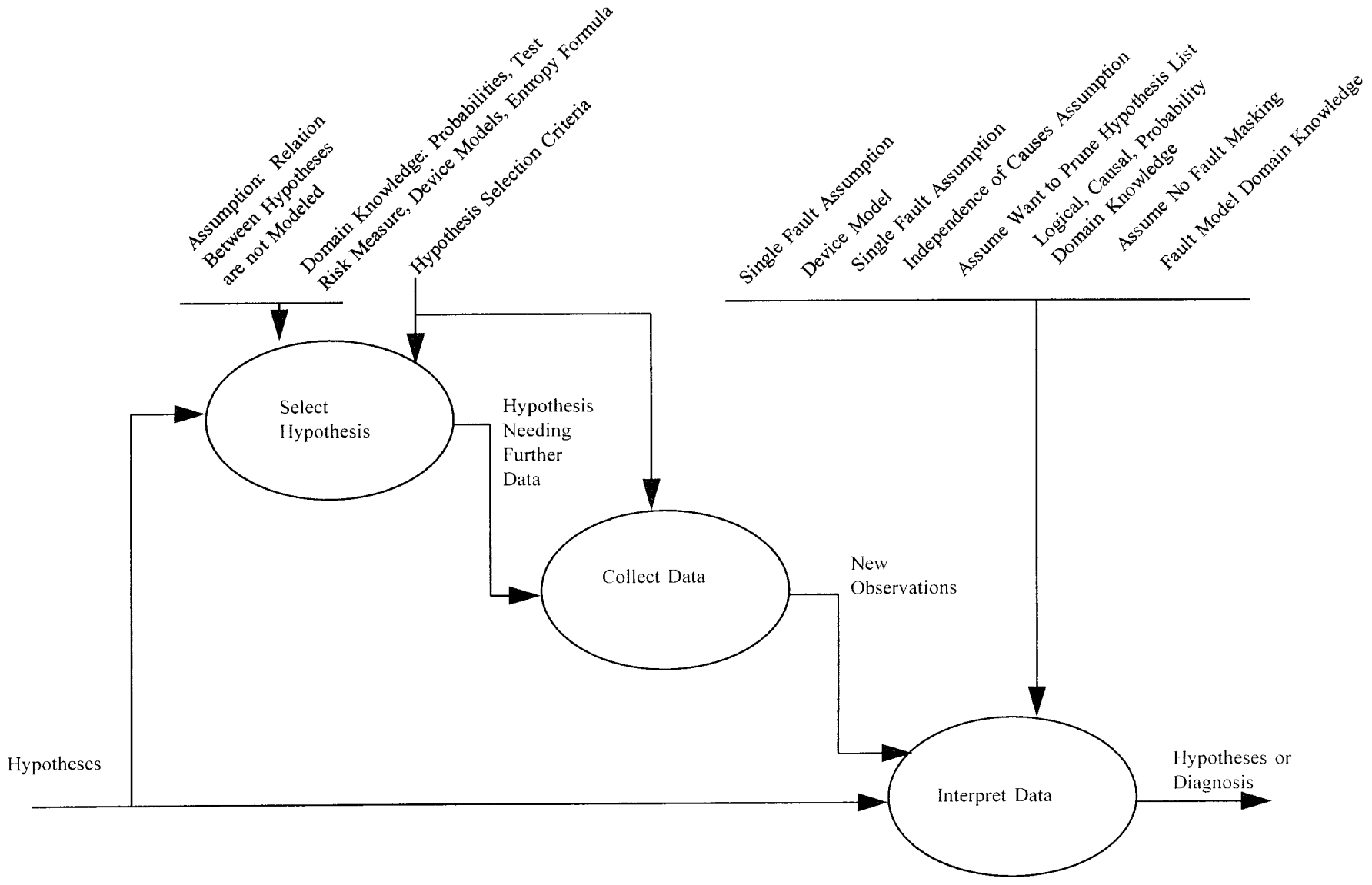


Fault Simulation

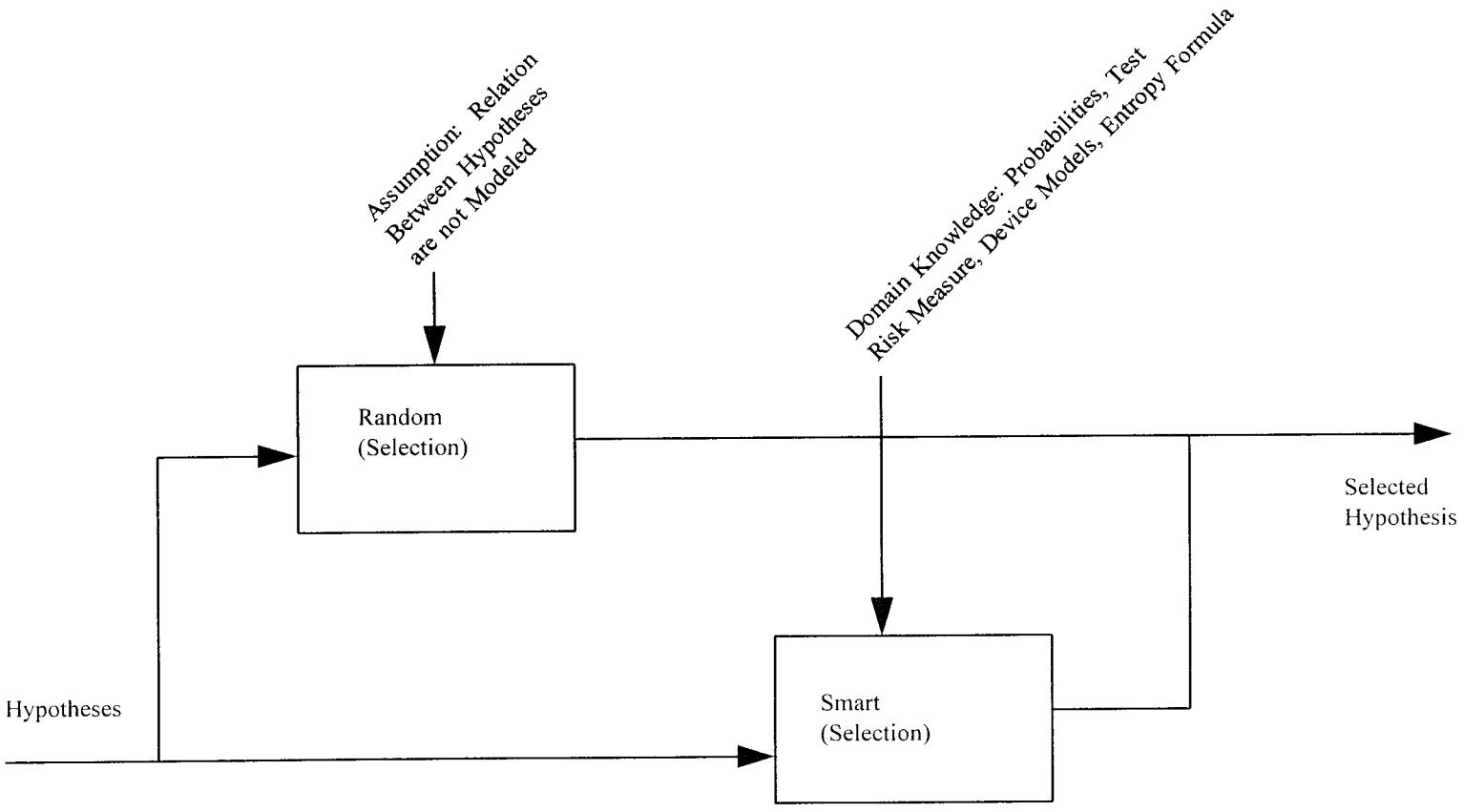

Assumption: Relation Between Hypotheses are not Modeled
Domain Knowledge: Probabilities, Test Risk Measure, Device Models, Entropy Formula
Single Fault Assumption
Hypothesis Selection Criteria
Device Model
Single Fault Assumption
Independence of Causes Assumption
Assume Want to Prune Hypothesis List
Logical, Causal, Probability Domain Knowledge
Assume No Fault Masking
Fault Model Domain Knowledge
Hypotheses
Hypothesis Discrimination
Hypotheses or Diagnosis



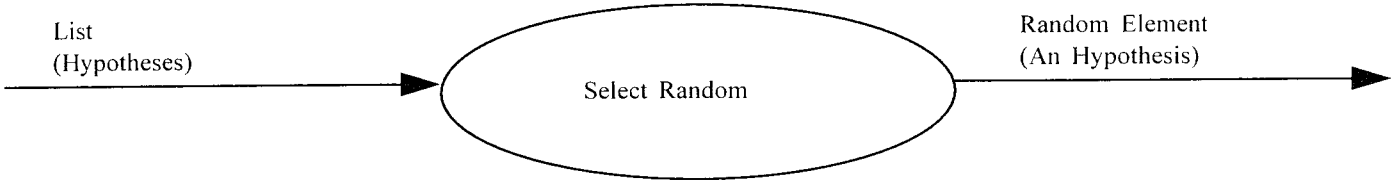
Hypothesis Discrimination



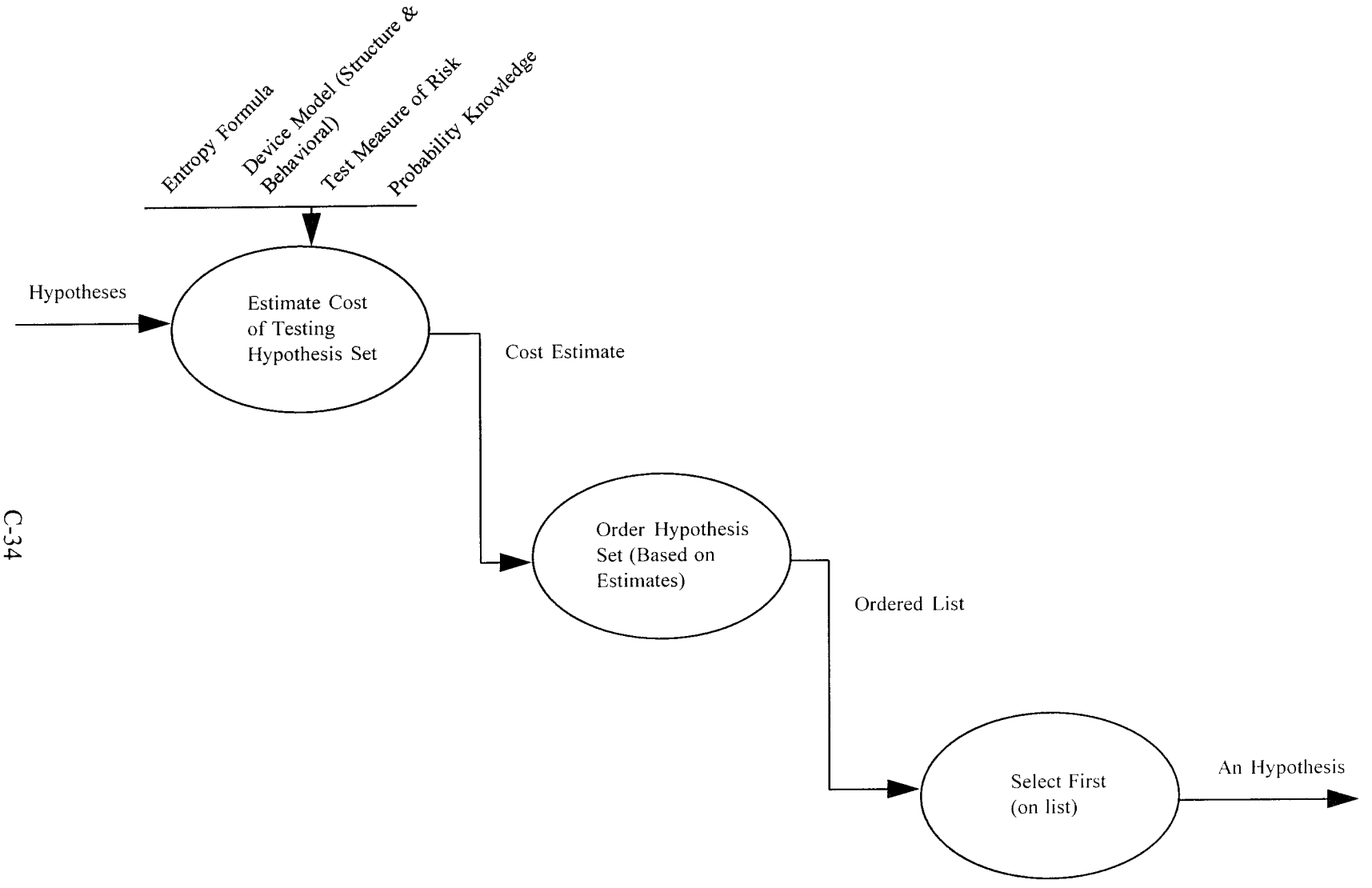
Discrimination



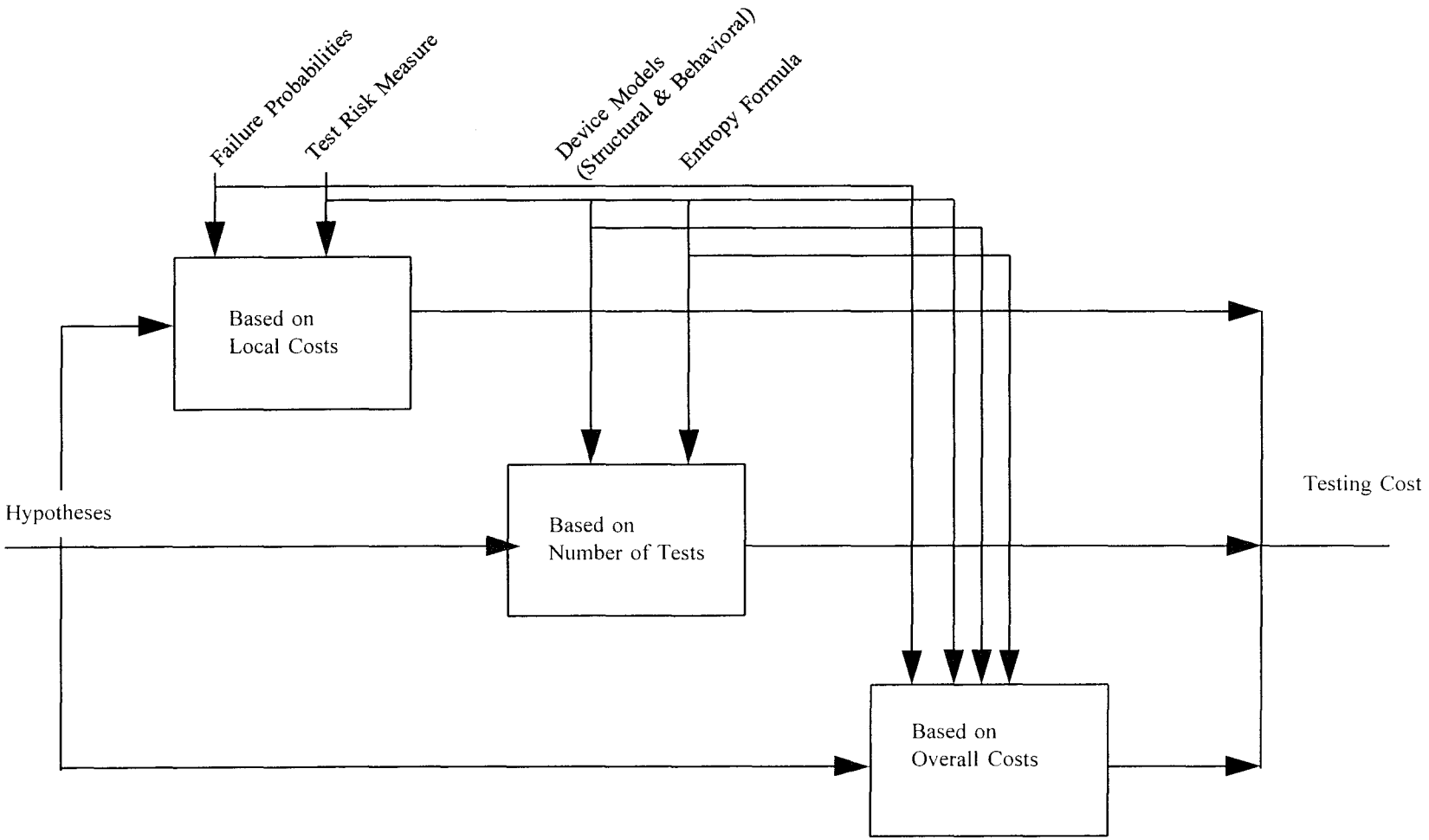
Select Hypothesis



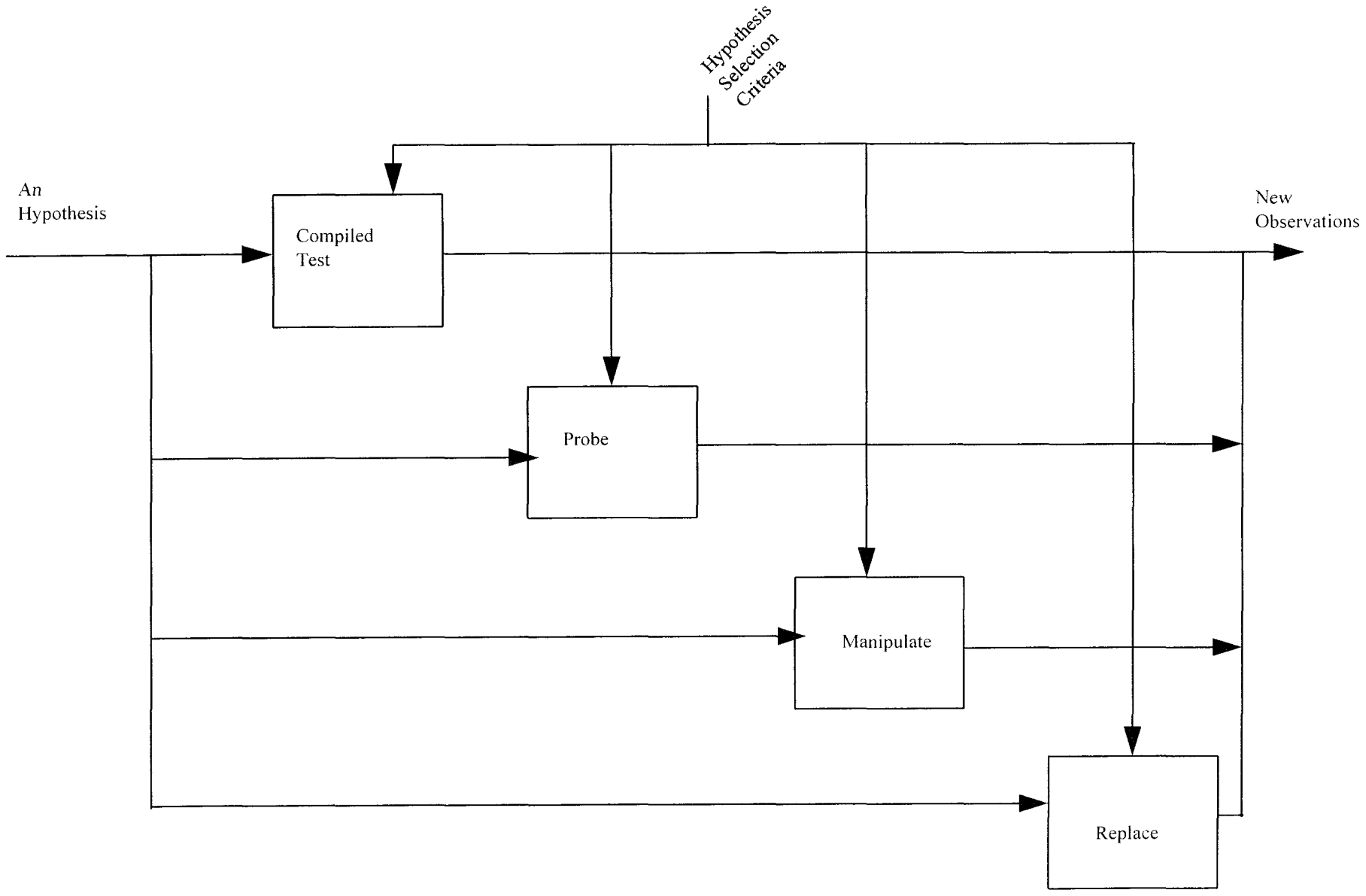
Random (Selection)

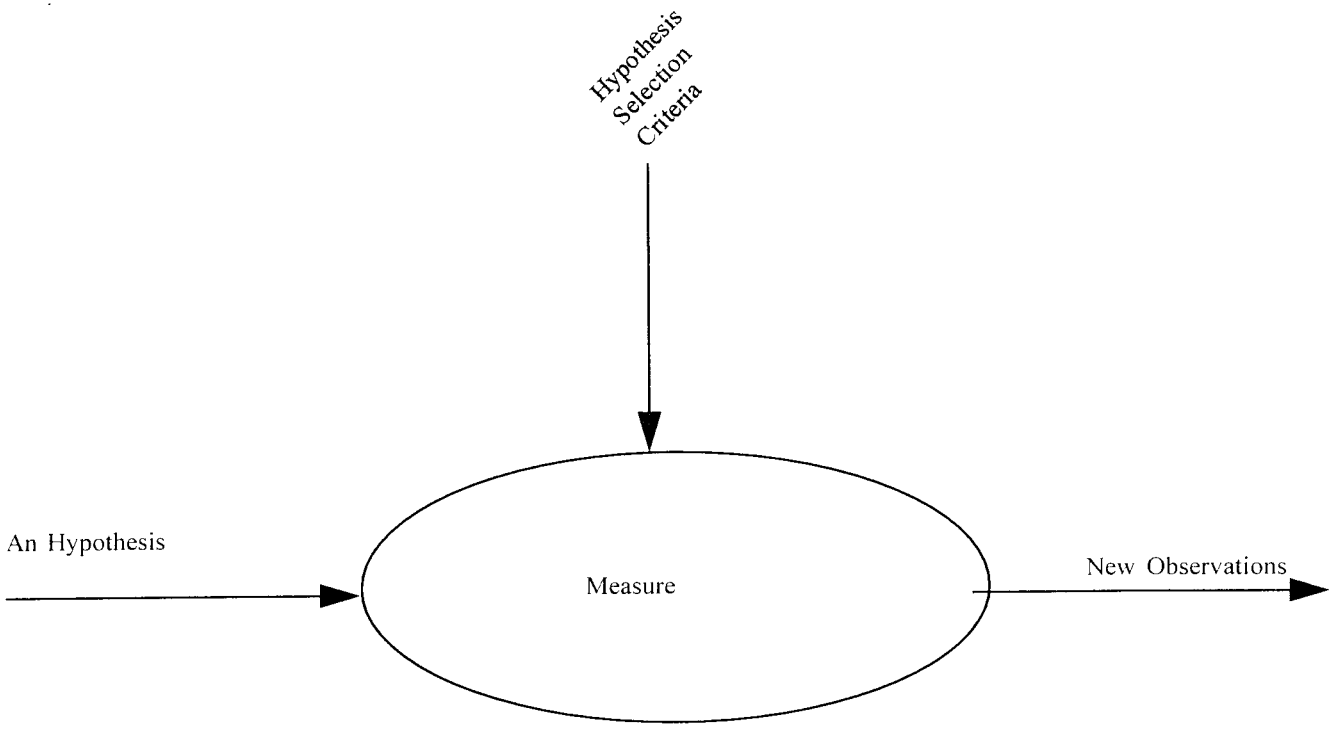


Smart (selection)

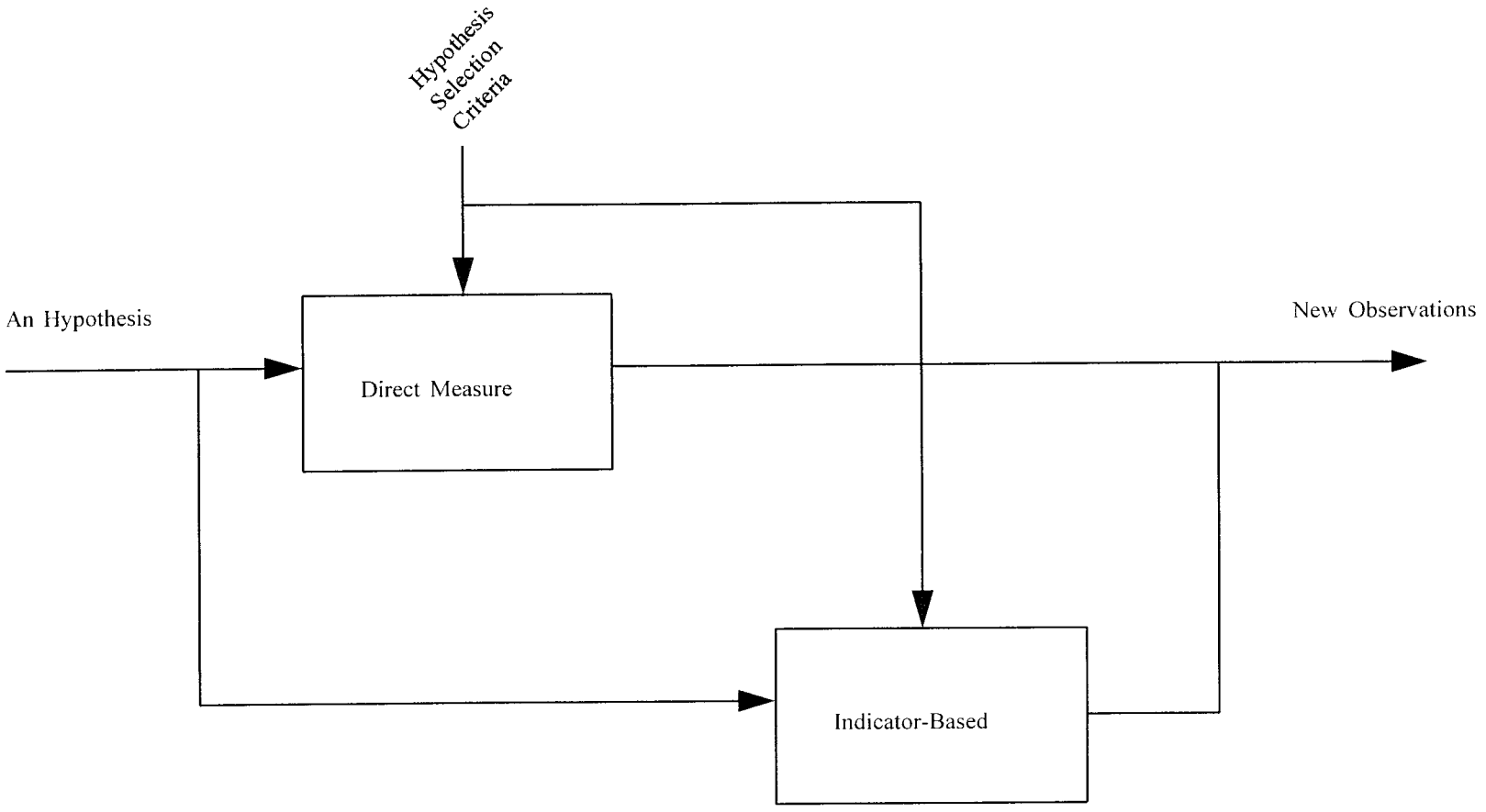


Estimate Cost of Testing Hypothesis Set

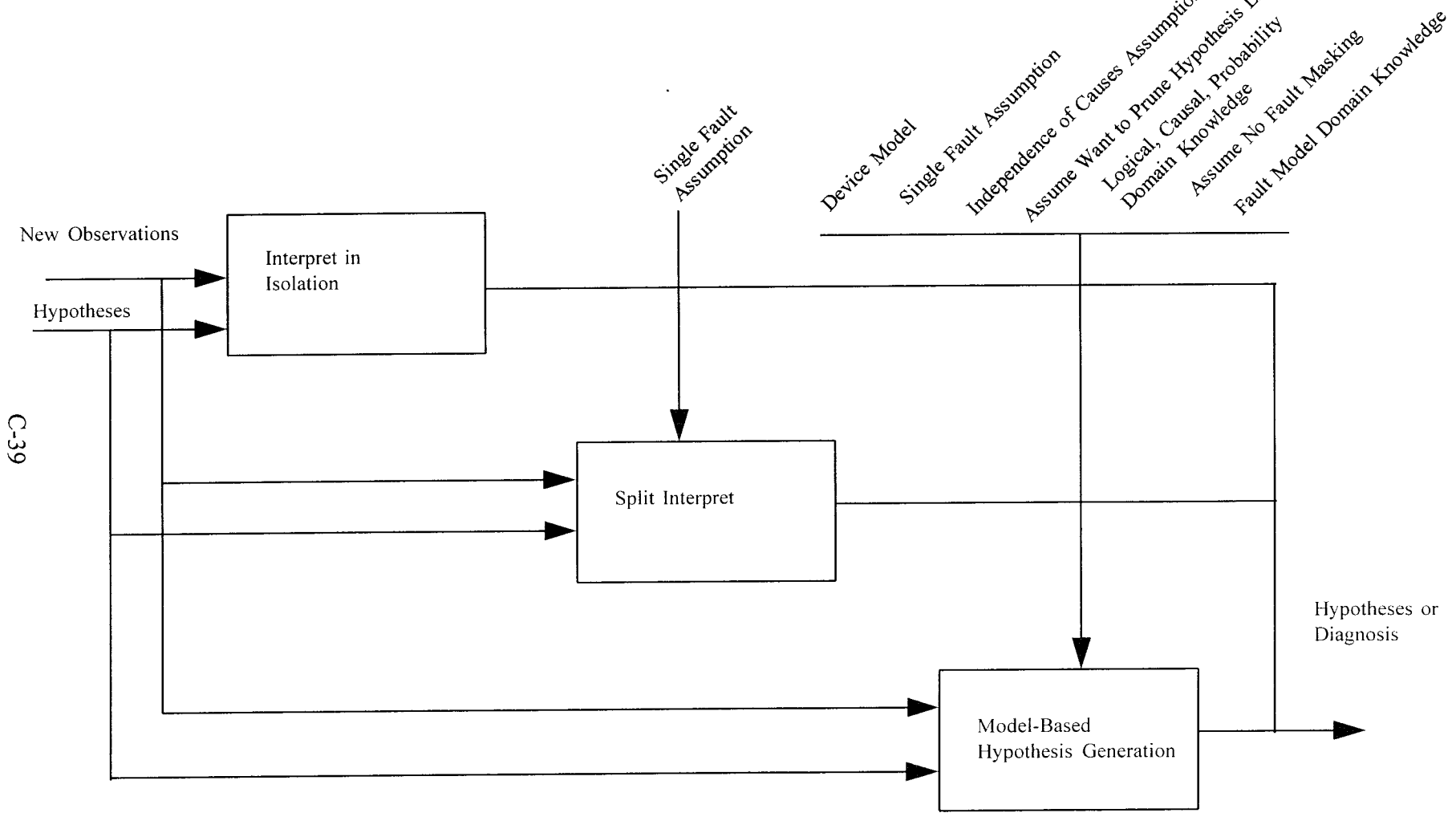

Hypothesis
Selection
Criteria
An
Hypothesis
Compiled
Test
New
Observations
Probe
Manipulate
Replace
Collect Data


Hypothesis Selection Criteria
An Hypothesis
Measure
New Observations

Probe


Hypothesis Selection Criteria
An Hypothesis
Direct Measure
New Observations
Indicator-Based

Measure



Interpret Data

Bibliography

1. Benjamins, Richard and Wouter Jansweijer. "Toward a Competence Theory of Diagnosis," IEEE Expert, 9: 43-52 (October 1994).

2. Benjamins, Richard and others. "Dynamic Method Selection in Diagnostic Reasoning," 12th International Conference of Artificial Intelligence, Expert Systems, and Natural Language. 153-164. Avignon France: 1992.

3. Chittaro, Luca and others. "Developing Diagnostic Applications Using Multiple Models: The Role of Interpretive Knowledge," in Industrial Applications of Knowledge-Based Diagnosis. Ed. Giovanni Guida and Alberto Stefanini. Amsterdam: Elsevier Science Publishers, 1992.

4. Console, Luca and others. "Abductive Diagnosis and its Application to a Mechanical Troubleshooting Problem," in Industrial Applications of Knowledge-Based Diagnosis. Ed. Giovanni Guida and Alberto Stefanini. Amsterdam: Elsevier Science Publishers, 1992.

5. Crowley, Nancy L, Deputy Chief Satellite Control and Simulation Division. "Satellite Control and Simulation Division: Software for a Complex World [The MAGIC Program]". Program Overview Briefing. Air Force Institute of Technology, Wright-Patterson AFB OH, 11 August 1994.

6. Davis, Randall. "Knowledge Acquisition in Rule Based systems: Knowledge About Representations as a Basis for system construction and Maintenance," in Pattern-Directed Inference Systems. Ed D. A. Watterman and F. Hayes-Roth. New York: Academic Press, 1978.

7. Davis, Randall and Walter Hamscher. "Model-Based Reasoning: Troubleshooting," in Readings in Model-Based Diagnosis. Ed. Walter Hamscher. San Mateo CA: Morgan Kaufmann Publishers, 1992.

8. Engelmore, R. S. and others. "Introduction," in Blackboard Systems. Ed. Robert Engelmore. Reading MA: Addison-Wesley Publishing company, 1988.

9. Finger, J. and Michael R. Genesereth. RESIDUE: A Deductive Approach to Design. HPP-83-46. Stanford CA: Stanford University Heuristic Programming Project, 1983.

10. Genesereth, Michael R. "The Use of Design Descriptions in Automated Diagnosis," in Readings in Model-Based Diagnosis. Ed. Walter Hamscher. San Mateo CA: Morgan Kaufmann Publishers, 1992.

11. Hamscher, Walter and others. "Preface," in Readings in Model-Based Diagnosis. Ed. Walter Hamscher. San Mateo CA: Morgan Kaufmann Publishers, 1992.

12. Harmelen, F. van and A. ten Teije. "Approximations in Diagnosis: Motivations and Techniques," Proceedings of the Dutch AI Conference (NAIC '95). Amsterdam: NAIC, 1995.

13. Harmelen, F. Van. Meta-Level Inference Systems. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1991.

14. Kelemen, Loretta A. Generic Satellite Monitoring Expert System. MS thesis, AFIT/GSO/ENG/94D-02. School of Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base OH, December 1994 (AAL-4649).

15. Leitch, RR and others. "A preliminary specification methodology for model-based diagnosis," in Annals of Mathematics and Artificial Intelligence, Vol 11. Ed. J. C. Baltzer. Basel Switzerland: Science Publishers, 1994

16. Mager, Robert F. Troubleshooting the Troubleshooting Course. Belmont CA: Pitman Learning, Inc., 1982.

17. Morris, William. The American Heritage Dictionary of the English Language. Boston: Houghton Mifflin Company, 1980.

18. Pearl, J. "Entropy, Information and Rational Decisions," Policy Analysis and Information Systems, 3: 93-109 (1979).

19. Punch, William F. A Diagnosis System Using a Task-Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. PhD dissertation. Ohio State University, Columbus OH, 1989.

20. Smith, Rauol. The Facts on File Dictionary of Artificial Intelligence. New York: Facts on File, 1989.

21. Stefanini, Alberto and others. ARTIST Final Report. ESPRIT Project P5143. Milan Italy: CISE, June 1993 (Document Number CISE 7838).

22. Struss, Peter. "An Amalgamation of Model-Based and Heuristic Reasoning for Diagnosis," in Industrial Applications of Knowledge-Based Diagnosis. Ed. Giovanni Guida and Alberto Stefanini. Amsterdam: Elsevier Science Publishers, 1992.

23. Struss, Peter. "Diagnosis as a Process," in Readings in Model-Based Diagnosis. Ed. Walter Hamscher. San Mateo CA: Morgan Kaufmann Publishers, 1992.

24. Taie, M. "MYCIN," in Encyclopedia of Artificial Intelligence. Ed. Stuart C. Shapiro. New York: Wiley-Interscience Publications, 1987.

25. Teije, A. ten and F. van Harmelen. "Using Reflection Techniques for Flexible Problem Solving, with Examples from Diagnosis," Proceedings of the IJCAI '95 Workshop on Reflection and Metalevel Architecture and their Applications in AI. Montreal: IJCAI, 1995.

VITA

John E. Friskie [redacted]a. He graduated from Derry Area Senior High School in June 1985 and entered the University of Pittsburgh as an Air Force ROTC scholarship cadet later that year. In April 1989, he received his undergraduate degree of Bachelor of Science in Electrical Engineering. Upon completion of the Communications-Computer Systems Officer course at Keesler AFB, MS, he was assigned to Hanscom AFB, MA. At Hanscom he was project manager for software development in the REACT (Rapid Execution and Combat Targeting) ICBM Launch Control Center Upgrade System Program Office, then was transferred to be project manager for software development and chief test manager for the B-2 Mission Planning System Program Office. At the B-2 SPO, he was responsible for leading the test team which successfully qualified both the functional and operational requirements of the mission planning software block which flew with the first operational B-2A stealth bomber. After attending Squadron Officer School, he entered the Graduate Computer Engineering program, Artificial Intelligence sequence, at the School of Engineering, Air Force Institute of Technology, in June 1994.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Master's Thesis

4. TITLE AND SUBTITLE

AN ARCHITECTURE FOR DYNAMIC META-LEVEL PROCESS CONTROL FOR MODEL-BASED TROUBLESHOOTING

5. FUNDING NUMBERS

6. AUTHOR(S)

John E. Friskie, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/95D-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Nancy L. Crowley, Lt Col, USAF
PL/VIQ, 3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

There are numerous methods used for troubleshooting devices. Each method has certain domains, knowledge requirements, and assumptions required for it to perform well. However, oftentimes no one method by itself is sufficient to completely solve a troubleshooting. Therefore, an architecture is required to control the combined use of many problem solving methods. The combination of multiple problem solving methods makes the troubleshooting process more robust in terms of device domains that can be dealt with and quality of diagnoses produced.

Troubleshooting has two tasks: diagnosis and problem resolution. This research provides an architecture that allows dynamic method selection during diagnosis. Dynamic method selection factors the current state of the diagnosis process along with other method parameters to determine which method to use to advance the diagnosis process.

The architecture was developed by combining themes from diagnosis research that focused on dynamic multimethod diagnosis and its control.

This work has produced several results. It provides an architecture to organize the methods and a basis for making control decisions concerning method use during diagnosis. It identifies a generous number of methods useful to perform diagnosis. It identifies the knowledge these methods require.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, Computer Aided Diagnosis, Expert Systems, Meta-Level Inference, Model-Based Reasoning	140
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL